TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

OPTICKÉ ROZPOZNÁNÍ ZNAKŮ NA FPGA OBVODU

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 3906T001 – Mechatronika
Autor práce: Bc. Daniel Kohout
Vedoucí práce: Ing. Martin Rozkovec, Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

OPTICAL CHARACTER RECOGNITION ON FPGA

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 3906T001 – Mechatronics
Author: Bc. Daniel Kohout
Supervisor: Ing. Martin Rozkovec, Ph.D.

Liberec 2014

Tento list nahraďte

originálem zadání.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé diplomové práce Ing. Martinu Rozkovci, Ph.D. za věnovaný čas, cenné rady a vedení. Dále bych chtěl poděkovat své rodině za vytvoření studijního i materiálního zázemí při vypracování mé diplomové práce i v průběhu studia.

Abstrakt

Cílem této diplomové práce je vytvořit v obvodu FPGA zapojení, které dokáže z vloženého obrazu s textem rozpoznat znaky. V první části práce je popsána teorie a historie samotného rozpoznávání (OCR) a druhy příznaků. Práce se dále zaměřuje na to, jak je možné jednotlivé příznaky u rozpoznávaných znaků rozdělovat. V další části je popsána také historie Matlabu, který byl využíván pro simulaci a odladění rozpoznávacího algoritmu. Ve třetí části jsou popsány použité programy pro přípravu neuronové sítě a také program na hledání a vyřezávání a optimalizaci znaků. Dále je popsána tvorba neuronové sítě a její efektivnost na dalších vzorcích. Ve čtvrté části je popsána samotná deska ZedBoard a prostředky k jejímu nastavení a naprogramování. V páté části je popsán návrh a nahrání obvodu do FPGA a jeho spuštění. V poslední části je navržený obvod nahrán do desky ZedBoard, kde jsou provedeny simulace. V závěru jsou zhodnoceny a porovnány všechny výsledky do tabulky.

Klíčová slova: FPGA, OCR, Matlab, neuronová síť, Xilinx Zynq, System generator

Abstract

The aim of this thesis is to create a circuit in FPGA which can identify text with signs from the image. The first part describes the theory and the history of OCR and the types of attributes. The work is also aimed at how it is possible to classify the individual attributes of identified signs. In the next section is described the history of Matlab, which was used for the simulation and debugging of the recognition algorithm. In the third section are described the programs used for the preparation of a neural network and the programs for finding and cutting and optimization of the signs. Further is described the creation of the neural network and its effectiveness on other specimens. The fourth section describes the board ZedBoard and its setup and programming. In the fifth section is described the design of a recording circuit in FPGA and its execution. In the last part is the designed circuit uploaded into the ZedBoard where are performed the simulations. At the end are evaluated and compared all the results.

Keywords: FPGA, OCR, Matlab, neural network, Xilinx Zynq, System generator

Obsah

Seznam obrázků

Seznam Tabulek

1. Úvod

Důležitost zaznamenávat dění kolem sebe měl člověk už odpradávna. Jeho první snahy jsou vidět v jeskynních malbách. To bylo v době před dvaceti tisíci lety. Postupným vývojem, který trval sedmnáct tisíc let, se začaly znaky přetvářet do písmen, aby historici mohli zapisovat do kronik a deníků. V dnešní digitální době, kdy psané slovo se čím dále méně používá, jsme tlačeni psát jakoukoliv informaci v elektronické podobě. Proto dochází k rozvíjení rozpoznávání znaků, aby bylo možné již napsané texty digitalizovat. K tomu, aby mohlo k rozpoznávání vůbec dojít, je potřeba splnit podmínky kvality textu. Je jasné, že text, který je potřeba převézt do digitální podoby, musí být naskenovaný kvalitně, v dobré kontrastu a pod správným úhlem.[5]

Cílem této diplomové práce je funkční návrh rozpoznávacího algoritmu v FPGA obvodu. Oskenovaný text se nahraje do vnitřní paměti FPGA a algoritmus zjistí, o jaká písmena se jedná. Tento rozpoznávací algoritmus je řešen pomocí neuronové sítě. Do neuronové sítě se nahrává určitá metoda pro rozpoznávání znaků na základě určitých příznaků např. perceptronu. Proto pro každý znak existuje v neuronové síti jeden neuron, který rozpoznává pouze jeden konkrétní znak. Jednoduše řečeno se jedná pouze o násobení a sčítání a o porovnání výsledku s určitou konstantou.

Takto testovaný znak projde přes celou škálu neuronů, ale pouze u správného neuronu je na výstupu jednička. U všech ostatních neuronů je na výstupu nula. Celý algoritmus byl nejdříve simulován pomocí programu Matlab. Po simulacích a odladění je vše vytvořeno jako hardwarový obvod v Systém generatoru, který byl dále nahrán do desky ZedBoard. Poté byly provedeny další simulace, aproximace a zhodnocení měření.

2. Optické rozpoznávání znaků – OCR

Cílem rozpoznávání znaků je převod naskenovaného textu do digitální formy. Původ zkratky pochází z angličtiny Optical Character Recognition. Jedná se o optické rozpoznávání znaků. U optického rozpoznávání může docházet k chybám, které vznikají nekvalitním skenovaným textem, nedostatečným rozlišením nebo nekvalitním vytištěním. Dalším problémem je podobnost některým znaků. Pokud nejde OCR provést, jedinou možností je přepsat text ručně. Pro rozpoznávání se využívá mnoho komerčních i nekomerčních programů. V tabulce (Tabulka 1) níže jsou uvedeny některé z nich.[8][14]

Tabulka 1 - Software pro OCR

Software pro rozpoznávání znaků	Dostupnost
ABBYY FineReader OCR	Komerční
Adobe Acrobat	Komerční
PDF-XChange Viewer	Freeware
PDF-XChange Viewer Pro	Komerční
GOCR GPL	Open source
Microsoft Office Document Imaging	Komerční
NovoDynamics VERUS	Komerční
Ocrad GPL	?
OCRopus Apache	?
OmniPage	Komerční
Readiris	Komerční
ReadSoft	Komerční
SimpleOCR	Freeware a komerční
SmartScore	Komerční

2.1 Historie OCR

Počátky OCR sahají do roku 1929, kdy si Gustav Tauschek nechává patentovat přístroj na rozpoznávání znaků. Do samotné stavby stroje se však pouští až roku 1933. Jedná se převážně o mechanický stroj, který funguje na principu přikládání jednotlivých šablon. Shoda šablony se znakem byla vyhodnocována pomocí fotoreceptoru.

Další vývoj v této oblasti přinesl krypto-analytik David H. Shepard, který pracoval na přepisu dat do strojové formy. Společně s kolegou Harvem Cookem se pokusili sestavit přístroj, který si po roce patentovali pod názvem „Aparát na čtení“. Další jejich přístroj „Gismo“ slavil větší úspěch. V té době již měli založenou firmu Intelligent Machines Research Corporation. Jejich přístroj pomohl v rozvoji rozpoznávání znaků a možnosti komerčního využití. Toho si všimla i společnost IMB, které projevila zájem o jejich vynálezy a patenty, které později odkoupila.

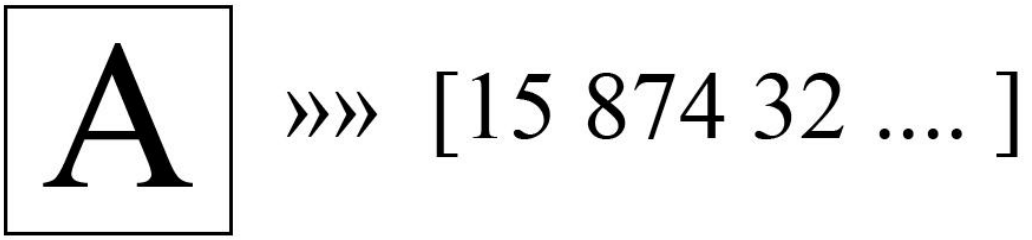
O další rozvoj se IMB pokusila samostatně. Zadávala dalším malým firmám tvorby přístrojů na rozpoznávání znaků. V této době se také objevuje samotná zkratka OCR. I když se v současné době již opticko-mechanický přístup nepožívá, tak se tato zkratka používá do dnes.

2.2 Příznaky znaků

Z pohledu rozpoznávání je důležité správně zvolit příznaky, podle kterých budou znaky rozeznávány. Příznak je soubor informací, které jsou unikátní pro každý znak. Většinou je tento soubor informací uložen do vektoru. Sady příznaků vznikají z předem připravených vzorů. Sady jsou dále optimalizovány, aby měly všechny stejný rozměr. Pro každý znak existuje mnoho zdrojů příznaků, které vznikají tak, aby byly od sebe co nejodlišnější. Čím větší rozdíly, tím lepší budoucí rozpoznávání. Existují však znaky, které jsou si velmi podobné. Proto i jejich příznaky jsou si velmi podobné. Příznaky se tedy musí zkombinovat tak, aby byl výsledek co nejlepší. Většina příznaků je vytvářena co nejjednodušeji, ale existují i složitější výpočty pro určování příznaků. V dalších podkapitolách jsou předvedeny příklady příznaků.

2.2.1 Převedení obrazu na vektor

Základním příznakem je převod obrazu na vektor. Díky tomuto převodu se získává kompletní informace o obrazu, která však obsahuje i různé nedůležité a nadbytečné části. Pro odstranění nadbytečných těchto částí se využívá matematický postup zvaný *„Fisher's linear discriminant"*, který redukuje dimenze dat tak, aby zůstaly jen nejdůležitější informace, ale v této práci použit nebyl.

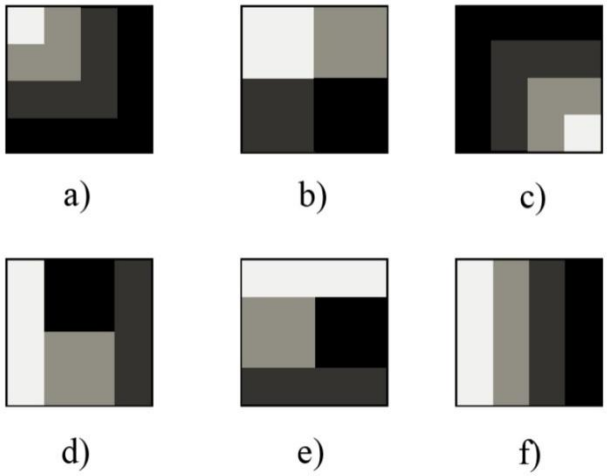


Obr. 1 - Převod obrazu na vektor

Tento příznak byl naprogramován pomocí dvou *for* cyklů. Matice samotného obrázku se převádí do nové proměnné ve formě vektoru. Zdroj obrázku se nachází v proměnné **k.** Výstupný vektor se nachází v proměnné **priznak1**. Jedná se o jednoduchý postup. Program vytvořen v prostředí Matlabu je uveden v příloze A na CD pojmenovaný *„priznaky.m"*.

2.2.2 Rozdělní obrazu na samostatné kvadranty a výpočet poměru

Tento druh příznaku vychází z kontrastu v jednotlivých částech znaku. Jednoduše se určí počet černých nebo bílých bodů v dané ploše znaku. V tomto případě byly vytvořeny čtyři kvadranty v různých pozicích znaku, z důvodu získání co nejvíce odlišných informací. Výsledkem toho příznaku je opět vektor, který obsahuje dvacet čtyři hodnot.



Obr. 2 - Rozdělení obrazu na kvadranty

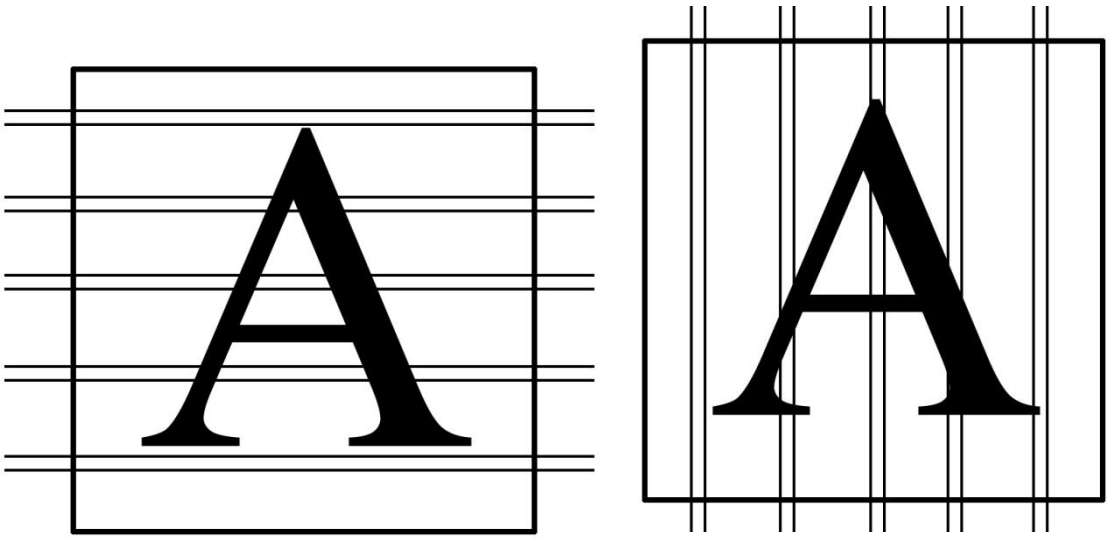
Programově je tento příznak opět vytvořen ze dvou *for* cyklů, které prochází celý znak. V každém kvadrantu je funkce *if*, která porovnává hodnotu černé nebo bílé. Následně se akumuluje do proměnné. Pro každý kvadrant je vytvořena samostatná proměnná. Všechny proměnné jsou poté uloženy do společného výsledného vektoru. Program je v příloze A na CD pojmenovaný *„priznaky.m“*.

2.2.3 Určení průchodů přes určité části znaku - horizontálně a vertikálně

Dalším jednoduchým typem příznaku je výpočet počtu průchodů znakem. Znamená to, že je znak v určitých částech procházen a hledá se určitá členitost znaku, například u písmena **A** (Obr.3). Při prvním horizontálním průchodu není nalezena žádná hodnota. Dalším průchodem je nalezen jeden průchod přes znak. Ve třetím průchodu je nalezena shoda ve dvou částech. Takto se prochází celý znak a určuje se počet průchodů. Ten je poté uložen do vektoru a dále se pracuje již s celým příznakem. Pro zajištění vyšší přesnosti se využívají i vertikální průchody.

Program jednoduše prochází po obraze po předem dané trase a hledá znak. Pokud narazí na barvu, uloží si do proměnné jedničku. Poté program čeká do doby, než narazí na

další prázdnou bílou plochu. Od té doby opět hledá další možný průchod barvou, aby proces opakoval a zapsal a přičetl hodnotu do proměnné. Počet průchodů přes obraz se nastavuje ve *for* cyklu, kde první proměnná určuje počet průchodů a druhý určuje směr procházení (shora dolů nebo zleva doprava). Program je uveden v příloze A na CD pojmenovaný *„priznaky.m“*.



Obr. 3 - Horizontální a vertikální průchody přes znak

2.2.4 Určení průchodů přes obraz pod úhlem

Dalším velice podobným příznakem je průchod přes obraz pod určitým úhlem. Jedná se o obdobný postup jako u předchozího příznaku, jen je průchod veden pod úhlem 45°. Liší se pouze v pohybu po znaku.

Obr. 4 - Průchod přes obraz pod úhlem 45°

Program je navržen pro tři průchody přes znak. Směr průchodů je z levého horního rohu obrazu do rohu pravého dolního. Jako první je počítána diagonála, poté průchod nad ní a pod ní. Využívá se zde rovnosti výšky a šířky znaku, která je brána jako jedna proměnná **x_celkem**. Pro každý průchod přes obraz je použit jeden *for* cyklus, který běží v mezích od jedné do hodnoty v proměnné x_celkem. Pokud při průchodu přes obraz program narazí na hodnotu nula a předchozí hodnota se nule nerovná, tak se výsledek akumuluje do proměnné. Tímto způsobem vzniká příznak. Tento program je také obsažen v příloze A na CD pojmenovaný *„priznaky.m“*

2.2.5 Další možné příznaky

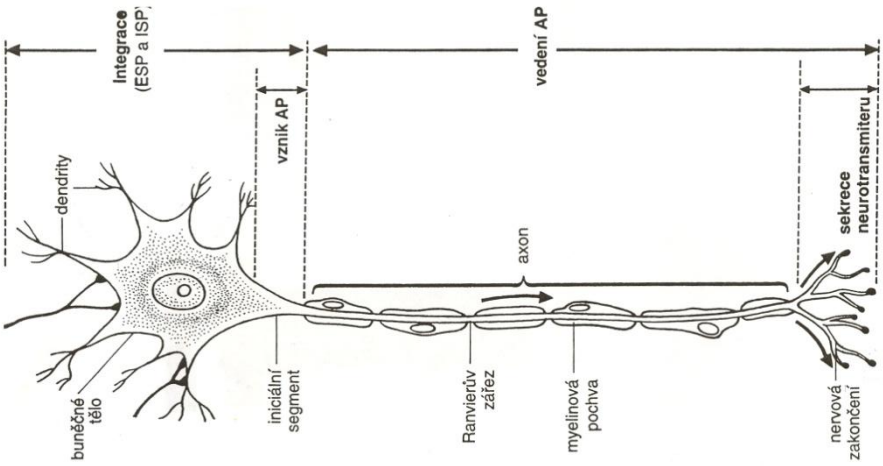
U předešlých příznaků byl znak prahován pouze na černou nebo bílou barvu. Existuje však celá řada příznaků, které pracují v celém spektru. Jedná se například o výpočty energií pro řádky nebo sloupce, transformace a sériové expanze, rozčlenění bodů, strukturální analýzy a další složitější možností je výpočet Fourierovy transformace. Dále je možné vytvářet příznaky dle jasů a jejich kontrastu v obraze. Další příznak by mohl vzniknout z histogramu obrazu.

3. Neuronová síť

Neuronové sítě pocházejí z modelu struktur neuronů v živých organismech. Jejich hlavní schopností a předností je schopnost učit se (někdy se i doučovat). Proto se využívají v zařízeních s umělou inteligencí. Existují dva směry výzkumu neuronových

sítí. Za prvé je to pochopení, jak funguje lidský mozek, a za druhé jsou to neurofyzické poznatky, které inspirují stojní inženýry.[4]

Neurony jsou v přírodě buňky v živých organismech, které umožňují vést signály a reagovat na ně. Jeden samotný neuron se skládá z mnoha dendritů, které přijímají signál z okolí. Tyto dendrity jsou navzájem propojeny. Další částí neuronu je axon, který je schopen samotný signál vyslat. Rychlost přenosu informace je v rozmezí 5-125m/s. Nervová soustava člověka obsahuje 10^{11} – 10^{12} neuronů. Postupem věku se počet neuronů snižuje. Odumře-li velká část neuronů, lze tuto ztrátu částečně nahradit zvýšením počtu spojů – dendritů.



Obr. 5 – Struktura neuronu[10]

Shrnutí činnosti neuronů: Neurony zachycují signály pomocí dendritů. Tyto signály se šíří dále do buňky, kde vzniká potenciál. Je-li tento potenciál dostatečně velký, neuron je schopen sám vyslat signál dál.

3.1 Historie neuronových sítí

První práce s neurony se datují k roku 1943, kdy s prvním jednoduchým modelem neuronu přišli pánové Warren McCulloch a Walter Pittse.[9]

Další pokrok v této oblasti přinesla kniha ***The Organization of Behavior*** od Donalda Hebba, kterou napsal roku 1949. Jednalo se o pravidla pro použití synapse neuronů.

Inspirací mu bylo pozorování reflexů u různých živočichů. V roce 1957 zobecnil model neuronu Frank Rosenblatt na takzvaný ***perceptron***, který počítal s reálnými čísly. *„Jedná se o pevnou architekturu jednovrstvé sítě s "n" vstupními a "m" výstupními neurony. Zároveň navrhl učící algoritmus, který v konečném čase nalezne odpovídající váhový vektor parametrů nezávisle na počáteční konfiguraci. V tomto případě se reálné stavy neuronů ve vstupní vrstvě nastavily na vstup sítě a výstupní neurony počítaly svůj binární stav, který určil výstup sítě stejným způsobem jako obecný neuron."*[9]

Dalším velmi podobným neuronovým prvkem je ***adaline***. Rozdíl tohoto modulu spočívá ve výstupu ze sítě, který je realizovaný lineární funkcí. Bernard Widrow a jeho studenti byli autory **ADA**ptivního **LIN**eárního **E**lementu. Sám Widrow se zasloužil o vznik první komerční organizace v polovině 60. let, která se zaobírala stavbou neuropočítačů a jejich součástek.

Karl Steinbuch v přelomu 50. a 60. let vyvinul ***model binární asociativní sítě.*** *„Na rozdíl od paměti klasických počítačů, kdy klíč k vyhledání položky v paměti je adresa, u asociativní paměti dochází k vybavení určité události či informace na základě její částečné znalosti. Rozvoj zaznamenaly dva typy asociativních pamětí a to* ***autoasociativní a heteroasociativní****. U prvního druhu šlo o upřesnění vstupní informace a u heteroasociativní dochází k vybavení určité sdružené informace na základě vstupní asociace. Rozdíl byl, že místo afinních kombinací počítal jen lineární kombinace vstupů. Biasy byly nulové, jedná se o označení jednoho ze vstupu do neuronu, který má nastaven váhu na záporný vnitřní potenciál.*" [9] Přes řadu nesporných úspěchů jsou sítě a neuropočítače používány pouze z experimentálního hlediska. [9]

V 80. letech se objevuje řada vědců, kteří zkouší dále neuropočítače využít. Díky jejich snaze opět toto téma ožívá a dostávají ho do popředí. Důkazem toho jsou tzv. Hopfieldovy sítě, které fungují na principu autoasociativní paměti. U této sítě jsou všechny neurony výstupní.

V roce 1986 David Rumelhart, Geoffrey Hinton a Ronald Williams publikují učící se algoritmus zpětného šíření chyby tzv. backpropagation pro vícevrstvou síť. Podaří se jim vyřešit problémy, které pozastavily vývoj. Tento algoritmus je jedním z nejvyužívanější a je požíván v 80% všech neurosystémů. Tento model vznikl zobecněním sítě perceptronů pro acyklickou architekturu se skrytými vrstvami.

Díky vyřešení problémů ve stavbě neuronových sítí, se opět tato oblast začala zkoumat. Výsledkem toho byla konference roku 1987 v San Diegu zaměřena výhradně na neuronové sítě. Setkalo se zde přes 1700 účastníků a zakládá se mezinárodní společnost pro výzkum neuronových sítí (INNS - International Neural Network Society). Od toho roku se po celém světě na univerzitách zakládají nové ústavy s tímto zaměřením.

V dnešní době se především výpočty simulují na klasických PC stanicích. Některé nadějné neurosítě jsou zaváděny do praxe, ať již v podobě automatického řidiče nebo náhrada u měřících přístrojů. Hopfieldova metoda se dá velice jednoduše použít na rozpoznávání obrazu.

3.2 Druhy neuronových sítí

Neuronové sítě se dělí dle struktury do dvou hlavních skupin. První jsou sítě s dopředným šířením signálu. Druhá skupina je se zpětnou vazbou. V dnešní době se většinou používají sítě se strukturou dopředného šíření signálu, kdy výstup signálů z jedné vrstvy je přiveden na vstup další vrstvy. Konečné výstupy jsou výstupy z celé sítě. Na rozdíl od dopředného šíření signálu se struktura se zpětnou vazbou liší v tom, že výstupy z jednotlivých vrstev jsou vedeny zpět na dané vrstvy. Díky tomu je možné realizovat výpočty založené na iteračním procesu, a tak řešit například optimalizační úlohy.[4]

Neuronové sítě s dopředným šířením se dále dělí opět do dvou skupin a to podle funkce, kterou realizují. Dělí se na lineární a nelineární. Výsledná funkce jednoho neuronu není totožná s funkcí celé sítě. Lineární sítě jsou schopné realizovat lineární matematické funkce a jejich skládání např. sčítání a násobení. Jednoduchou logickou funkcí je AND, která je realizována pomocí jednoho neuronu.

Hlavní vlastností nelineární neuronové sítě s dopředným šířením signálu je schopnost učit se. Vlastní učení je první fáze, která slouží k určení váhových koeficientů a uložení informací do paměti systému. Samotné učení probíhá dvěma způsoby a to s učitelem nebo bez něj. Při učení s učitelem je síť trénována pomocí připravených dvojic vstupních dat a je přesně očekáván výstupní vzorek. Vstupní vzorek musí být z množiny vybrán takový, aby splňoval všechny důležité vlastnosti množiny pro danou úlohu. Zde jsou nenatrénované síti vloženy vstupní vzorky. Podle skutečných odezev na vstupní data se mění váhový koeficient. Během tohoto trénování se na vstup přivedou všechny

vzorky vícekrát a náhodně. Síť musí po natrénování správně reagovat na všechny vzorky z množiny. Aby byla síť co nejpřesnější, musí být počet vzorků co nejvyšší. Přetrénovanost sítě není na závadu, jedná se jen o zbytečný čas navíc.

Druhá varianta učení bez učitele je realizována pouze s trénovacími vzorky a výstupní vzorek neexistuje. Výstupní vzorky přísluší k jednotlivým vstupním vzorkům během procesu učení. Důležité váhové koeficienty se nastaví tak, aby pro každý vstup byl jediný aktivní výstup. Díky tomu při přivedení vzorku na vstup se aktivuje jen jeden jedinečný výstup.

4. Prostředí Matlab

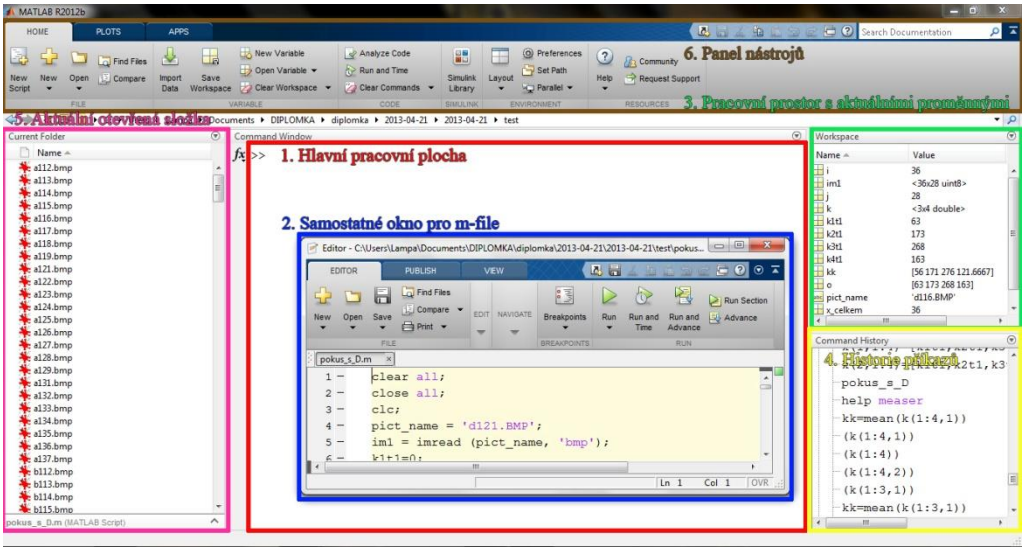
Matlab, neboli **mat**rix **lab**oratory, což volně přeloženo znamená maticová laboratoř. Jedná se o programové prostředí od společností MathWorks. Má mnoho funkcí a možností: výpočty s maticemi, vykreslování 2D i 3D grafů funkcí, implementaci algoritmů, počítačovou simulaci, analýzu a prezentaci dat i vytváření aplikací včetně uživatelského rozhraní. Skutečnost je taková, že hlavní výpočty se provádějí pomocí matic. Původně byl Matlab určen pro matematické účely, ale časem byly přidány nové funkce a rozšíření a tím se stalo jeho využití velmi široké. Využívají ho především vědeckotechničtí pracovníci, studenti, zaměstnanci vysokých škol. Využití nachází i v soukromém sektoru. Nejčastěji se používá v technických oborech a ekonomii.[2]

Samotný program byl vytvořen koncem sedmdesátých let profesorem Cleverem Mollerem, který působil na Univerzitě v Novém Mexiku. Důvodem bylo, aby studenti mohli využívat Linpack a Eispack aniž by se museli učit programovací jazyk Fortran. Jelikož byl Matlab jednodušší a srozumitelnější, jeho obliba rostla. Rozšířil se na další univerzity, kde byl využíván především pro matematické účely. V roce 1983 se začal o Matlab zajímat Jack Little, který viděl jeho využití pro ekonomické účely. Jeho zájem přerostl ve spolupráci na jeho vývoji, přepsal Matlab do jazyka C a přidal další funkce a knihovny. V roce 1984 založili Little, Moller a Bergert společnost MathWorks a začali vydávat Matlab již komerčně. Do této doby byl zdarma. Další vývoj postupoval v souladu s vývojem počítačů. Jelikož ze začátku jejich výpočetní kapacity nebyli tak velké, mohl Matlab pracovat pouze omezeně. Široké možnosti využití se ukázali při

příchodu počítačů s virtuální pamětí, které sice zpomalili dobu výpočtu, ale jejich množství a proveditelnost už omezena nebyla.

Jak již bylo řečeno, veškeré proměnné se ukládají do vektorů (matic), se kterými Matlab dále pracuje. Rozměr těchto matic je tvořen podle potřeby. Datové typy, se kterými umí Matlab pracovat, jsou například: celočíselné, s plovoucí desetinou čárkou, komplexní, znaky, struktury, které mohou obsahovat další matice, symbolické a další speciální. Velkým rozdílem oproti jiným „programovacím prostředí“ je ukládání a deklarace proměnných. V Matlabu se deklarace datového typu provede automaticky nebo se musí přesně zadat. Během existence proměnné se může libovolně datový typ měnit. Což je výhoda, ale i nevýhoda.

Samotné programování v Matlabu se provádí přímo na hlavní ploše nebo pomocí **m-filů**. Jedná se o skripty, které postupně posílají příkazy na hlavní obrazovku. Takto lze vytvořit funkce, které se dají zpětně vyvolávat a zjednodušit tak samotnou práci. M-fily se dají ukládat, kopírovat, duplikovat a pracovat s nimi i mimo matlab (zjednodušeně jedná se o textové soubory). Pokud se zadá v m-filu příkaz zakončen středníkem (;), je pouze proveden, ale v hlavním okně se po spuštění informace neobjeví. pokud středník uveden není, informace se vypíše i s výsledkem na hlavní pracovní plochu.



Obr. 6 - Popasné okno Matlabu

1.Hlavní pracovní plocha

2. Samostatné okno pro m-file

3. Pracovní prostor s aktuálními proměnnými

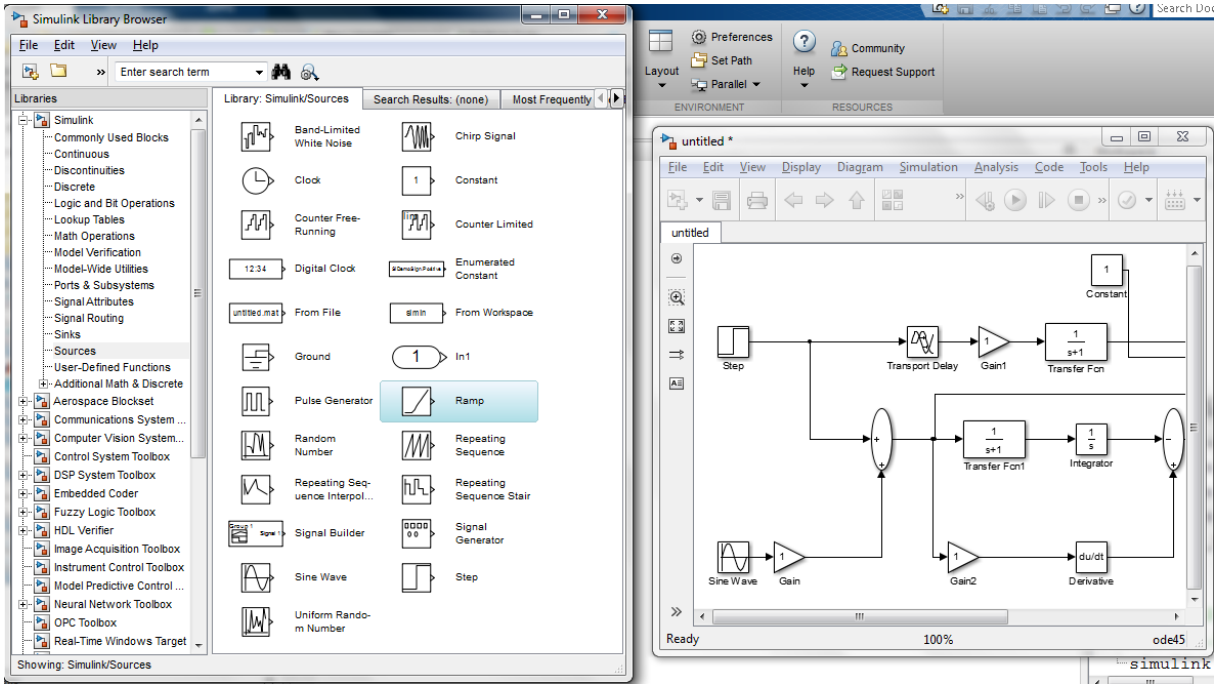
4. Historie příkazů

5. Aktuální otevřená složka

6. Panel nástrojů

Další již zmíněnou důležitou částí Matlabu je **Simulink**. V tomto prostředí je možné vytvářet návrhy různých typů např. diferenciální rovnice. Výhodou je, že návrhy lze vytvářet i bez znalostí programování. Jedná se pouze o spojování vybraných bloků do schémat. Jejich průběh je dle potřeby časově omezen.

Jako vstupy se dají využít proměnné z pracovního prostoru Matlabu, nebo je lze nastavit přímo v Simulinku jako konstanty. Výstupy mohou být posílány do pracovního prostru Matlabu nebo je lze zobrazovat přímo v Simulinku pomocí bloku *scope*.



Obr. 7 - Okno simulinku

4.1 Algoritmus pro vyřezávání znaků z textu

Prvním krokem při vytváření neuronové sítě je příprava vzorků znaků. Proto byl naprogramován algoritmus, který toto vyřezávání řeší. Jeho prácí je vyříznout znaky z připraveného oskenovaného obrazu a následně uložit do složky. Dále je možné se znaky pracovat. Program je uveden v příloze A na CD pojmenovaný *„vyrezavani.m“*.

Jako první je provedeno načtení *imread* oskenovaného obrazu z knihy. Důležité je, aby byl v co nejvyšší kvalitě. Při skenování bylo nastaveno nejvyšší rozlišení a to 2400dpi.

Dalším postupem je převedení obrazu na stupně šedi a poté celkové vyprahování. Práh byl optimalizován ručně. Dále je v práci popsán algoritmus pro prahování jednotlivých znaků. Pokud je prahování dokončeno, spustí se příkaz *bwlabel,* který takto připravený obraz převede na matici, kde všechny spolu sousedící části mají stejnou hodnotu. To znamená, že matice má stejné rozměry jako obraz, ale obsahuje čísla od jedné do počtu částí v obrazu. Počet oblastí se také uloží do samostatné proměnné. Tato proměnná je otestována jednoduchou podmínkou, zda neobsahuje příliš budoucích znaků pro připravený zápis. Z jednoho obrazu lze vyříznout 8887 znaků. Tento počet je možný v případě potřeby zvýšit. Dále je proveden výpočet pozice těžiště pro každý znak.

Algoritmus prochází celý obraz a při nenulové oblasti uloží hodnotu umístění do proměnných. Pro každý samostatný znak vznikne nový řádek s počtem ploch určitého znaku. Poté je zkoumána každá oblast zvlášť a pomocí jednoduchého dělení se vypočítá těžiště každé oblasti.

Nyní, když už je znán počet oblastí a jejich těžiště, je možné začít s přípravou jednotlivých znaků pro vyřezávání. Vezme se těžiště zkoumané oblasti a od něj je vytvořena zkoumaná oblast o určitém rozměru. Aby u znaku nebyla zbytečná bílá místa, zkoumají se maxima ve všech čtyřech směrech od těžiště. Poté jsou procházeny oblasti mezi těmito maximy. Pokud má bod v této oblasti správnou hodnotu pro daný znak, uloží se do nové proměnné a vzniká tak samotný znak. Načítání nového znaku probíhá z původního obrazu (im). Existují však znaky, které obsahují dvě oblasti např. znaky **i** a **j**. U těchto znaků je vycházeno ze znalosti, že oblasti jsou nad sebou v určité vzdálenosti. Proto je i v programu u každé oblasti zkoumána příbuznost s další oblastí. Při nálezu se oblasti s maximy spojí a prochází se celá nová oblast a hledají se dané body pro znak.

Výsledný znak je uložen do proměnné *pismenko*. Dále je nutné ho pojmenovat a uložit do složky. Práce se soubory v Matlabu je jednoduchá, ale pro uložení je potřeba vytvořit název datového typu *char*. Jelikož není známo, jaký aktuální znak je, jsou znaky uloženy postupně podle čísel. Pro každou hledanou oblast se vytvoří samostatný název díky vytvořené inkrementaci.

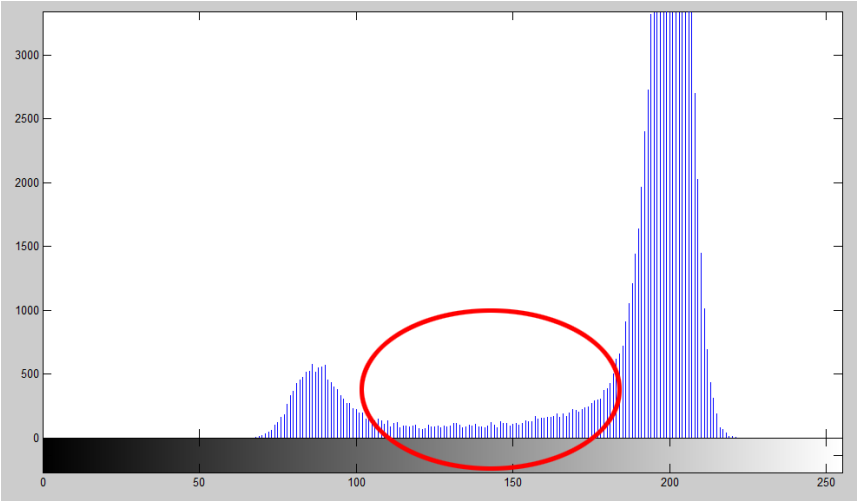
Proměnné *mm*, *m*, *k* a *l* představují proměnné číslo (například 1115). Číslo 46 je vyhrazeno pro proměnnou *jmeno* tečky a trojčíslí 98, 109 a 112 znamená *bmp*. Formát, ve kterém se bude znak ukládat, musí být uveden i v názvu. V poslední řadě se příkazem *imwrite* uloží aktuální znak do složky s přiřazeným jménem a typem formátu.

Další prácí s takto vyřezanými znaky je jejich rozdělení podle obsahu a optimalizace, aby byly všechny znaky nositeli stejné informace. Rozdělení bylo provedeno ručně. Optimalizace bude popsána v další kapitole.

4.2 Algoritmus pro prahování a optimalizaci znaku

Optimalizace znaku je přesné naprahování a nastavení přesné výšky a šířky. Prahování je důležitou součástí algoritmu, protože pro budoucí práci je potřeba převedení znaku na binární hodnoty (pouze 0 a 1). Nejdříve je ale potřeba převést znak na správnou velikost.

Prvně se načte znak pomocí *imread*. Poté se pomocí funkce Matlabu *imresize* změní velikost obrázku. Vnitřní fungování vychází z principu tzv. „nearest-neighbour“ (nejbližší soused). Do funkce je zadáno, co se má změnit a na jaké rozměry. Poté je na řadě prahování. Opět se využívá funkce Matlabu pro histogram, *imhist*. Princip nalezení prahu vychází z vyhledání lokálních minim v oblasti mezi dvěma vrcholy (znázorněná oblast v kroužku).



Obr. 8 - Ukázka histogramu

Algoritmus pracuje na základě jednoho *for* cyklu. Cyklus prochází hodnoty histogramu a mají-li hodnoty klesající tendenci a v určitém bodě se začnou zvětšovat, je bod uložen do proměnné *prah* jako hodnota prahu. Většinou se na rozsahu celého histogramu najdou 2-4 hodnoty pro práh. Jako hlavní je brána předposlední nalezená.

Poté již stačí projít celý znak a podle velikosti prahu porovnávat, zda budou hodnoty černé nebo bílé. V tomto případě se nastavují pro budoucí práci na hodnoty 0 nebo 1.

Nový znak je následně uložen stejným postupem jako u vyřezávání nebo se s ním dále pracuje přímo v dalších skriptech. Program je uveden v příloze A na CD pojmenovaný *„optimalizace.m“*.

4.3 Příprava příznaků pro neuronovou síť

Nyní když už jsou vzory znaků připraveny, roztříděny a optimalizovány je potřeba je ještě připravit jako příznaky pro neuronovou síť. Znaky byly roztříděny do složek a optimalizovány předešlým algoritmem. Všechny příznaky budou vytvářeny ze stejné sady znaků. Proto v programu se bude jen lišit výroba příznaku. Příklad přípravy sítě bude teda ukázán na příznaku převedení obrazu na vektor (kapitola 2.2.1).

Po zpuštění tohoto algoritmu musí vzniknout dvě matice. Jedna vstupní, označena jako **INPUTS**, bude obsahovat příznaky v řádcích pro každý znak. K této matici bude druhá **OUTPUTS**, která bude obsahovat na řádku jen jednu jeničku a to na místě kolikáté je

znak v pořadí v abecedě. Pro tuto síť byla vybrána jen malá písmenka (bez diakritiky – 26 písmen) a tedy počet vzorku písmenek je 4938. Což je minimálně 100 na jedno písmenko.

Samotný program na začátku načte informaci o názvech všech složek se znaky. Potom postupně prochází každou složku a každé písmenko. Vždy načte aktuální znak a převede ho na příznak a uloží do proměnné **INPUT.** Z ní poté se ukládá na přesně určený řádek do INPUTS. Hodnoty pro OUTPUTS vznikají hned pod ním. Jednička se nahraje do sloupce **(j-2)** což je číslo aktuálně otevřené složky. Počítá se od tří, jelikož příkaz dir na první dvě hodnoty rezervuje znaky teček.

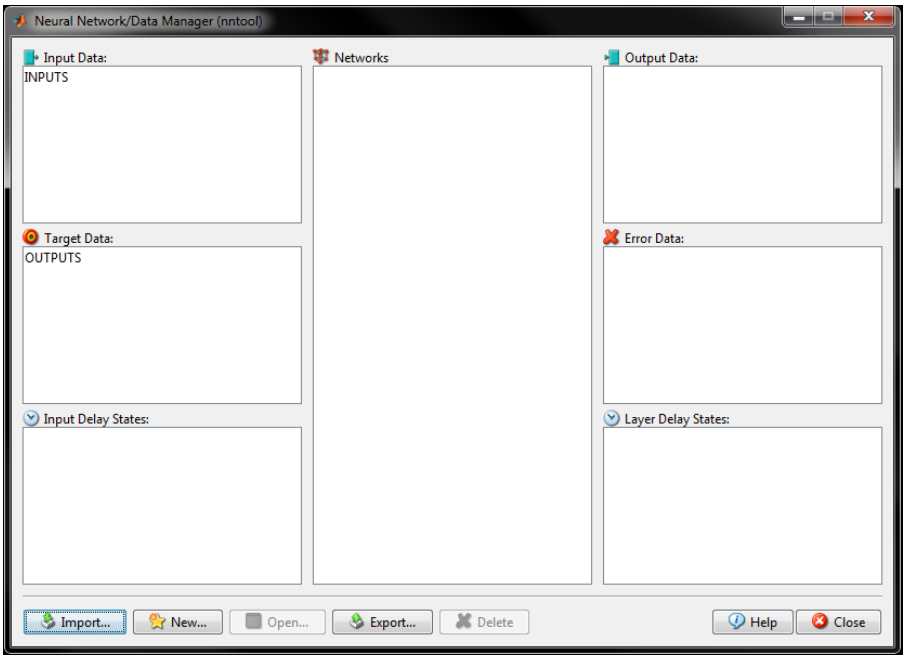
Na konci celého algoritmu je nutno transponovat matice pro další práci. Žádají si to vstupy do nástroje neuronových sítí v Matlabu. Program je uveden v příloze A na CD pojmenovaný *„priprava_pro_ns.m“*.

4.4 Vytváření neuronových sítí v programu Matlab

Program Matlab je vybaven speciálními příkazy a aplikacemi pro práci s neuronovými sítěmi a jejich vytváření. Existují tři základní postupy, jak v Matlabu vytvořit neuronovou síť. První z nich je pomocí aplikace neuronové sítě. Ta je umístěna v záložce APPS. Kompletní vytvoření neuronové sítě je prováděno pomocí turialu. Další možností vytvoření neuronové sítě je přímo v pracovní ploše a to pomocí příkazů. Ukázka příkazu je uvedena dále:

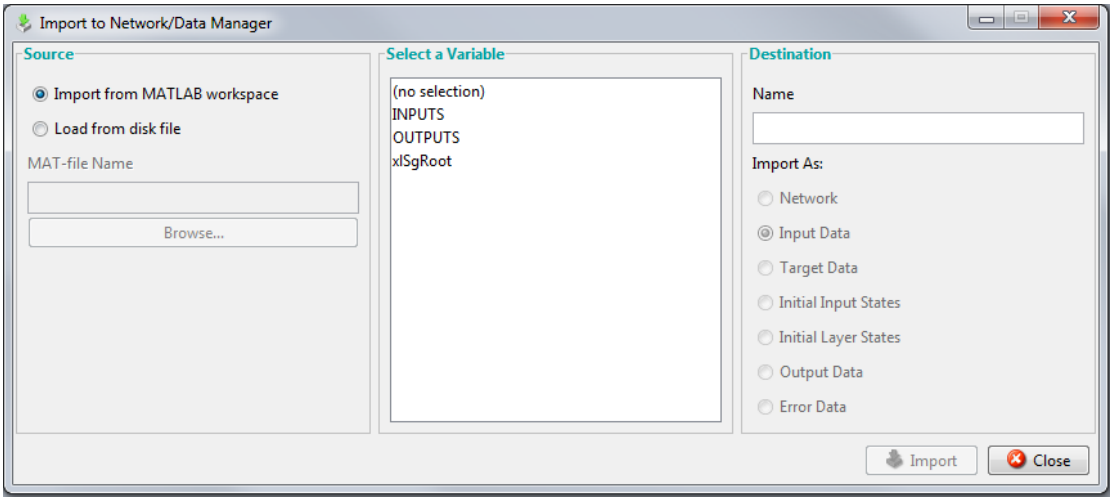
„`net=network(numInputs,numLayers,biasConnect,inputConnect,layerConnect,outputConnect)`“.

Další možností je použít příkaz *nntool*, který otevře speciální okno, kde se vytváří neuronová síť.



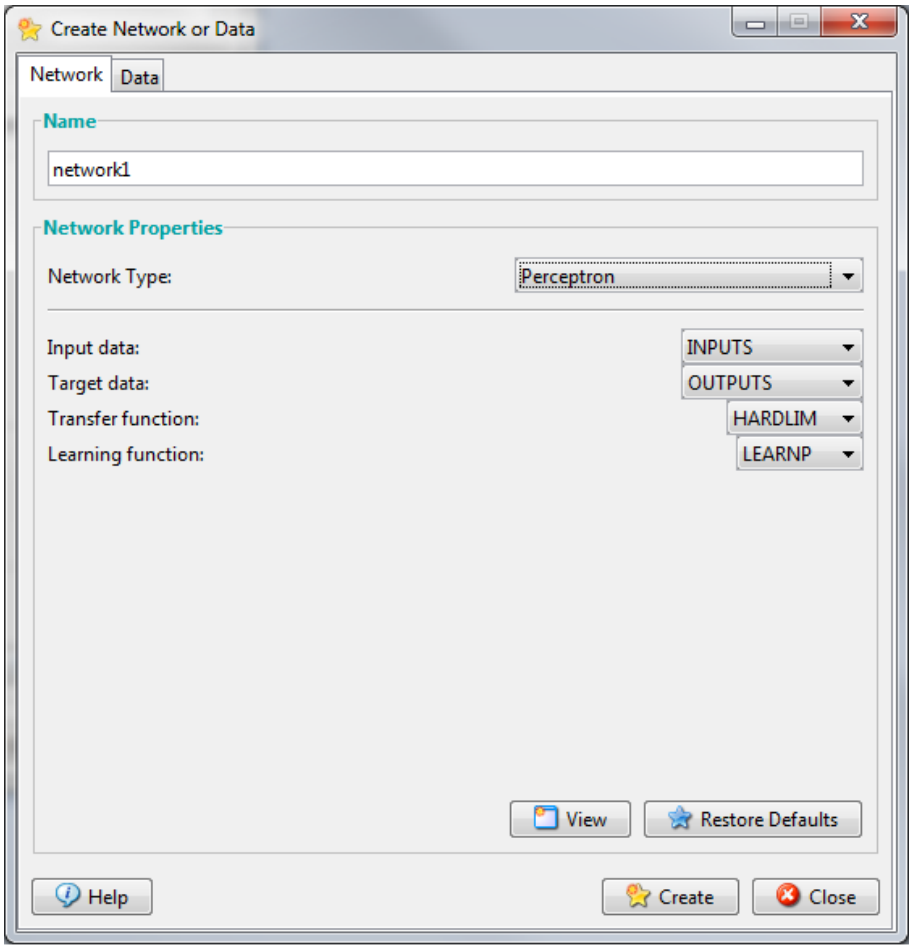
Obr. 9 - Okno nntool

Postup pro práci v tomto okně je následující. Jako první je potřeba pomocí tlačítka *Import...* vložit hodnoty pro vytvoření sítě (nebo vložit již připravenou síť). Data se dají načíst buď ze souborou (*-.mat) nebo přímo z matlabovské pracovní plochy. Jako vstupní data *Input Data* se načítají hodnoty vstupů pro trénink sítě. Dalšími načítanými daty jsou *Target data*, kterými jsou určeny výstupy z neuronové sítě.



Obr. 10 - Import to Network/Data manager

Po načtení se okno zavře a začne se vytvářet samotná síť. Tlačítkem *New...* se otevře okno, ve kterém se nastaví a vytvoří neuronová síť podle potřeby.



Obr. 11 - Okno pro vytvoření sítě

Jako první se nastaví název sítě. V dalším kroku je potřeba nastavit, jaký druh sítě se má vytvořit. Je zde mnoho možností např. Feed-forward backprop, Elman backprop, Hopfield, Perceptron, a další... . Pro každou variantu sítě jsou k dispozici další speciální záložky nastavení. Na příkladu bude předvedeno, jak vytvářet síť s percetronem. Po volbě typu sítě je nutné nahrát vstupní a výstupní data. V záložce *Input data* a *Target data* se vyberou připravená a nahraná data. Dalším nastavení je, podle jaké funkce se má síť převádět a podle které funkce učit. Po vytvoření sítě se okno zavře a samotná síť se otevře v okně *nntool*. Po otevření je síť zobrazena symbolicky. Přesunutím na záložku *Train* se dostaneme do oblasti, kdy se bude síť učit (trénovat). Zde se vyberou již použitá data a nastaví se názvy proměnných pro výstupní a chybová data. Poté se spustí samotný trénink *Train Network*. Otevře se další okno, ve kterém je zobrazen průběh tréninku. Po dokončení tréninku se uloží výsledky do připravených proměnný a je možno okno zavřít. Nyní je potřeba se zpátky vrátit do okna *nntool* a vytvořená data exportovat *Export...* a uložit buď do souboru *-.mat nebo do proměnné do prostoru

Matlabu. Data je možno využít a dále s nimi pracovat. Důležité je uložit si samotnou síť a zkontrolovat, zda je výstup chyb nulový.

Další práce se sítí je již v samotné prostředí Matlabu. Pro další práci je potřeba získat data vah z neuronové sítě. K tomu stačí si je nechat vypsat z uložené sítě a uložit. Výpis se provede tečkovou notací, což znamená název uložení sítě, tečka a příkaz IW(vahy_site=cell2mat(nazev_site.IW);). Dalšími potřebnými daty ze sítě jsou hodnoty pro porovnání výstupu z neuronu tzv. *vektor b* (vektor_b=cell2mat(nazev_site.b);). Postup je opět stejný, napíše se název sítě, tečka a b.

Nyní je již vše připraveno pro vytvoření návrhu v system generatoru. Pouze je potřeba zjistit, jaká síť a s jakými příznaky je nejoptimálnější.

4.5 Výsledky kvality neuronových sítí

Zkoušenými sítěmi byly Percetronová a Feed-forward backprop. Pro simulaci bylo prozkoušeno 725 nový znaků. V tabulce jsou uvedeny výsledky pro jednotlivé příznaky i jejich kombinace. Dále jsou napsány procentuální úspěšnosti a jsou napsány i orientační časy výpočtů jednotlivých sítí. Výsledná vybraná síť je uložena v příloze A na CD *„neurn_sit.mat“*.

Tabulka 2 - Porovnání příznaků a sítí

Příznak - pro síť Percetronu	**Doba výpočtu**	**Počet správně poznaných znaků**	**Procenta úspěšnosti**
2.2.1 Převedení obrazu na vektor	5-7hod	708	97,7%
2.2.2 Rozdělní obrazu na samostatné kvadranty a výpočet poměru	6-9hod	41	5,7%
2.2.3 Určení průchodů přes určité části znaku - horizontálně a vertikálně	6-9hod	35	4,8%
2.2.4 Určení průchodů přes obraz pod úhlem	6-9hod	22	3,0%
2.2.2 + 2.2.3	8-10 hod	73	10,0%
2.2.3 + 2.2.4	8-10 hod	38	5,2%
2.2.2 + 2.2.3 + 2.2.4	8-10 hod	95	13,1%
Všechny příznaky	7-9 hod	709	97,8%
Příznaky - pro síť Feed-forward backprop			
2.2.1 Převedení obrazu na vektor	3-5hod	695	95,9%
2.2.2 + 2.2.3 + 2.2.4	4-6 hod	88	12,1%

5. FPGA

Obvod FPGA je polovodičová součástka, jejíž funkci lze měnit na základě programového kódu. Tato její vlastnost vychází už z názvu FPGA - Field Programmable Gate Array (programovatelné logické pole). Použije-li se pro návrh systému programovatelné pole, návrh systému se obejde bez nutnosti použít obvody s

definovanou funkcí. Vlastnosti programovatelných polí dovolují měnit a přidávat funkce, přizpůsobovat hardware novým standardům apod. Obvody FPGA se řadí mezi skupinu obvodů, která je označována jako ISP – In System Programming, tedy programují se až po osazení na desku plošných spojů. V případě obvodů FPGA je tato vlastnost vyjádřena písmeny FP – Feild Programable. Obvodem FPGA lze realizovat jakákoliv logická funkce, podobně jako je tomu u zákaznických obvodů ASIC. Avšak oproti zákaznickým obvodům mají programovatelné nespornou výhodu, protože je kdykoliv možné jednoduše upravit jejich konfiguraci.[7]

Klasické programovatelné obvody jsou založené na vytvoření logické funkce pomocí matice propojení mezi elementárními logickými prvky. Obvody FPGA pracují na odlišném principu. Obsahují programovatelné logické buňky označované jako Logic Elements (LEs), což není nic jiného než přednastavené bloky paměti, ve kterých jsou uložené výsledky logických funkcí. Logické buňky jsou vzájemně pospojované strukturou maticových spojů, kterou lze programově měnit. Součástí obvodů FPGA jsou dále vysokorychlostní transceivery, vstupně/výstupní obvody a bloky logických obvodů. Propojením všech těchto částí je možné vytvořit široký škálu funkcí od základních, reprezentovanými logickými hradly AND, OR a XOR, až po složité kombinační funkce. Většina obvodů FPGA je uzpůsobena tak, že je možné paměťové buňky využít jako jednoduché klopné obvody nebo jako celé paměťové bloky.[7]

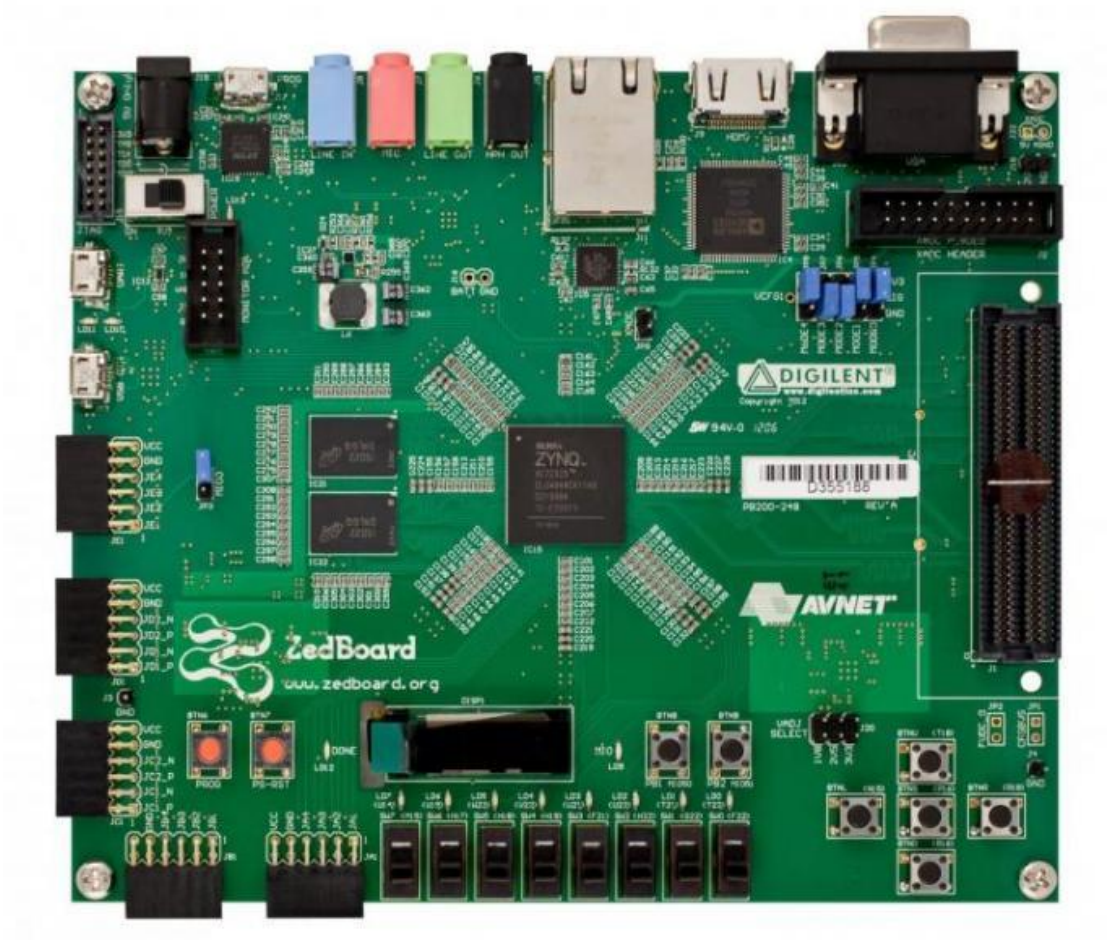
5.1 Architektura ZedBoard

Deska ZedBoar, která je osazena procesorem Zynq, pochází z velké rodiny od společnosti Xilinx All Programmable SoC architecture. Využívá integrovaný procesor dual-core ARM® Cortex™-A9. Zpracování systému je technologii 28nm. Srdcem celé desky je samotný centrický procesor Zynq. Okolo něj jsou rozmístěny ostatní součásti, jako například násobičky, sčítačky, čipy paměti a externí paměťové rozhraní a další. Dále je uveden základ, co vše se na desce ZedBoard nachází.[18][17]

Tabulka 3 - Základní vlastnosti desky ZedBoard[17]

Procesorová část (PS)	• Dual ARM® Cortex™-A9 MPCore™ @666 MHz • NEON™ Processing / FPU • 512 MB DDR3
FPGA část (PL)	• 13300 Slice (106400 Registrů, 53200 LUT) • 220 DSP bloků • 140 BRAM 36bitx1024
Propojení PS s PL	• 2xAXI-lite (32bit max. 400MB/s) • 4xAXI (64bit max. 1200MB/s)

Na obrázku je zobrazena deska ZedBoard s procesorem Zynq.



Obr. 12 - Deska ZedBoard[17]

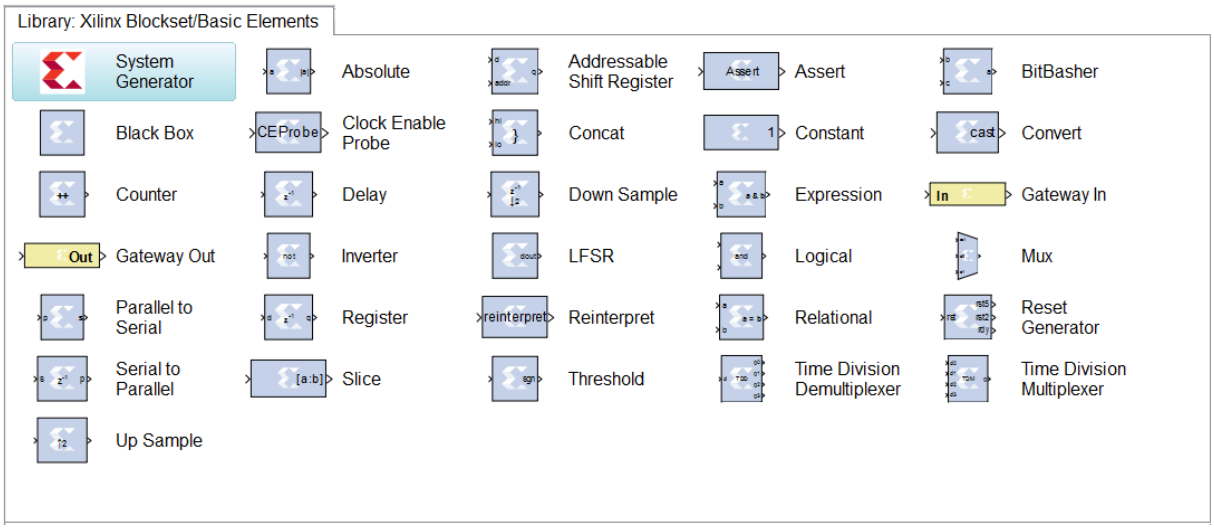
Velice důležité jsou vlastnosti AXI-lite. Návrh z Matlabu nedokáže pracovat na vyšším rozhraní, a proto jsou práce prováděny na tomto pomalejším typu. I maximální počet bitů, které je možné najednou zpracovávat, je omezen na 32. Rozhraní AXI-lite je podmnožinou AXI a je určené pro komunikaci s řídícími registry. Proto se musí

využívat i při použití vyššího řízení. Hlavním cílem AXI-lite je využívat jednoduché komponenty rozhraní a postavit z nich potřebný návrh. Díky tomu je jednodušší ověřit funkčnost.[3]

5.2 Prostředí System Generator

System Generator je vývojový prvek, který se začleňuje do programu Matlab. Jedná se o návrhové prostředí pro vytvoření zapojení na FPGA. V tomto návrhovém prostředí je možné navrhovat jak hardwarové součástky, tak i softwarové prvky. Kompletní System Generator je v Matlabu zobrazován v okně Simulinku. Po spuštění a založení nového modelu je nutné nastavit Simulink pro návrh hardwaru. Nastavení se provede přes Simulation -> Model Configuration Parameters (nebo Ctrl+E). Zde se změní nastavení *Type* z *Variable-step* na *Fixed-step*. Také se zde změní *Solver* (výpočtové zařízení) na *discrete*. Dále se v simulinkovém okně nachází Xilinx Blocket, ve kterém se schovávají všechny využitelné prvky pro návrh od jednoduchých až po komplexní. Je zde možné psát i vlastní m-kódy. Nachází se zde přes 100 bloků. Bloky jsou rozděleny do různých kategorií pro snadnější orientaci (například Basic Elements, Communication, Control Logic, Data Types, Math,…).

Mezi základní a nejčastěji využívanou kategorii patří **Basic Elements**. Dále jsou zobrazeny a popsány příklady základních bloků.



Obr. 13 - System generator Basic Elements

Mezi nejdůležitější bloky patří blok **System Generator**, který je nepostradatelný v každém návrhu. Nastavuje se v něm v jakém typu FPGA se bude kompilovat výsledný návrh. Díky němu je možné provádět různé simulace a převézt návrh do reálné podoby.

Dalšími důležitými bloky jsou **Gateway in** (vstup) a **Gateway out** (výstup). Díky nim lze do obvodu zavést informace (data) z připojených periferií. Na vstupu se nastavuje typ informace, který má být posílán, dále počet bitů a v jakém formátu má být s informací pracováno. Dělí se na Boolean (1/0), Fixed-point (pevně nastavená velikost posílaných bitů) a Floating-point (neurčená velikost posílaných bitů). Při volbě pevného nastavení se dále řeší, zda má být informace zasílána jako Signed (se znaménkem – přepočítáváno dvojkovým doplňkem) nebo Unsingled (bez znaménka). Poté se již nastaví pouze počet posílaných bitů.

Mezi další často používané bloky patří **Counter** (čítač). Jedná se o klasický čítač, který může inkrementovat nebo dekrementovat při určitém kroku. Rozsah konečného čítání lze nastavit, ale je zde možno nastavit, aby čítal po celou dobu práce obvodu. Jeho výstupní hodnota se nastavuje také jako počet poslaný bitů se znaménkem nebo bez něj.

Dalším velmi často používaným blokem je blok **Constant**. Jedná se o uložení konstanty, kterou můžeme využívat v návrhu. Její nastavení je stejné jako u Gateway In, jen s tím rozdílem, že číslo je pevně dané.

Další blok, který se používá, je **Delay** (zpožďovač). V tomto bloku se signál jen o určitou nastavenou dobu zpozdí. Využívá se pro synchronizaci, protože u některých matematických bloků dochází ke zpoždění.

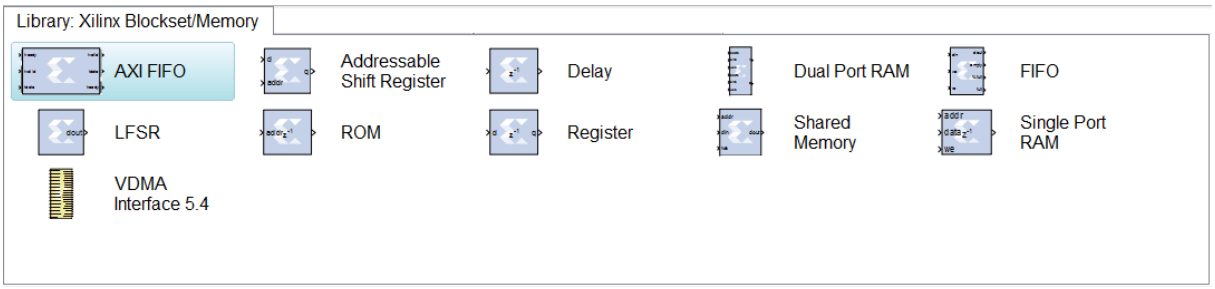
Mezi další bloky patří **Relational** (srovnávač). Jedná se o blok, který porovnává dvě příchozí hodnoty o stejném tvaru (stejný počet bitů). Porovnávání může být rovnost, nerovnost, větší a menší. Výsledkem je binární hodnota jedna nebo nula.

Pokud se spolu dvě hodnoty porovnávají, je nutné také výsledek dále využít. Proto existuje další důležitý blok a tím je **Mux.** Ten blok se skládá ze tří vstupů a jednoho výstupu. Vstup označený jako *sel* rozhoduje, jaký ze vstupů bude použit na výstupu, zda *d0, d1, d2, ...***.** V tomto bloku je možné nastavit počet vstupů.

Poslední dva často používané bloky z této kategorie jsou **BitBasher** a **Concat.** Blok BiBasher slouží jako rozdělovač bitů. To znamená, že se na vstup přivádí číslo ve správném tvaru, např. prvních pět bitů je zpracováváno odděleně a zbytek je

zpracováván na jiném kanálu také samostatně. Jednoduché naprogramování toho bloku bude ukázáno a vysvětleno v příkladech. Blok Concat pracuje zcela opačné. Na vstup přivedu určitý počet dat a výsledná hodnota na výstupu bude jen jediné číslo.

Další důležitou kategorií je **Memory** (paměti).

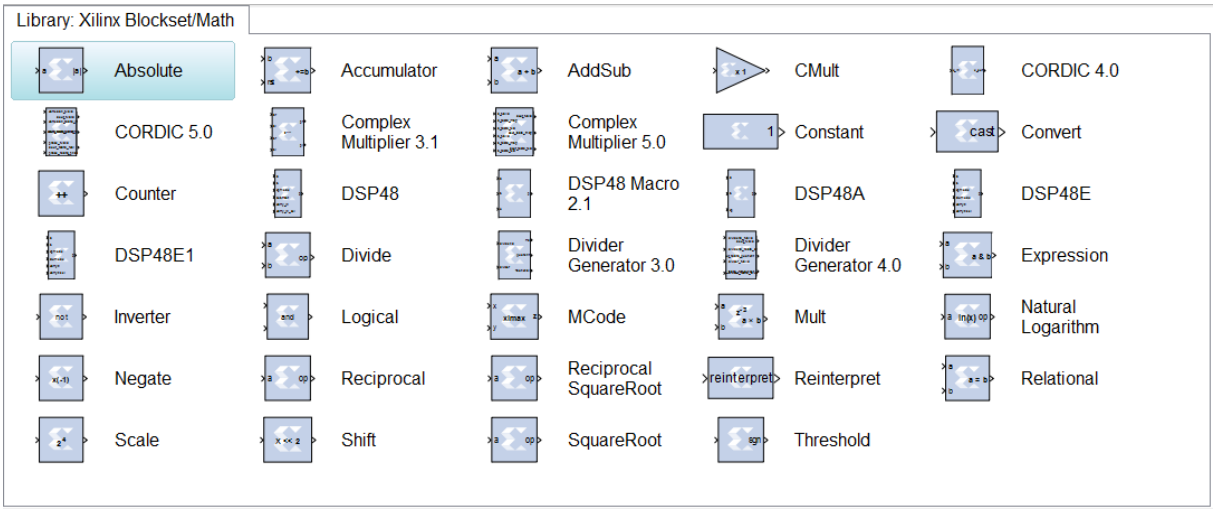


Obr. 14 - System generator Memory

Některé bloky z kategorie Memory již byly zobrazeny v kategorii Basic Elements, ale jsou zde také mnohé další. Nejčastěji využívaným blokem z této kategorie je blok **ROM** (pevná paměť), která je řízena vstupem a na výstup posílá vyvolanou hodnotu. Hloubka a obsah paměti se nastavuje přímo v bloku, kde se také nastavuje jako počet bitů a typ výstupní hodnoty, stejně jako u bloku Constant. V tomto bloku se také nastavuje, zda se jedná o distribuovanou paměť nebo o paměť RAM. Čtení z bloku ROM vnáší do návrhu zpoždění jedna.

Další často používaná paměť je blok **Dual Port Ram**. Tento blok může do sebe na rozdíl od bloku Rom zapisovat hodnoty, ale jeho natavení je obdobné. Tento blok dokáže také číst ze dvou pozic v paměti najednou. Pro každé čtení a zápis jsou tři vstupy a jeden výstup. Značí se A a B. Na vstup je přivedena adresa a data pro zápis. Na řídícím vstupu (we – write enable) se v daném okamžiku rozhoduje, zda zapisovat nebo ne.

Další důležitou kategorií je **Math** (bloky matematický operací).



Obr. 15 - System generator Math

Mezi nejdůležitější matematické bloky patří blok **Mult** (násobení). Jedná se o blok, kde jsou na vstup přivedena dvě čísla, které jsou spolu vynásobena. Výsledek je zobrazen s automatickým počtem bitů, ale je zde možnost i pevného nastavení. Doba výpočtu trvá tři cykly, proto tento blok vnáší do návrhu zpoždění tři.

Další blok z této kategorie je **AddSub** (sčítání). V tomto bloku se spolu dvě čísla sečtou nebo odečtou. Opět je zde možné nastavit, jak v jakém tvaru má být výstupní hodnota.

Dalším velmi důležitým používaným blokem je blok **Logical**. Jedná se o blok s logickými funkcemi, jako je AND, NAND, OR, XOR, NOR a XNOR. Nastaví se pouze počet vstupů a je opět možné pevně nastavit počet bitů na výstupu.

Existují ještě další kategorie a spousty dalších bloků. Zde je ukázáno pouze několik příkladů bloků, které jsou v system generatoru obsaženy, jejich možné využití a nastavení.

5.3 Xilinx Platform studio (EDK)

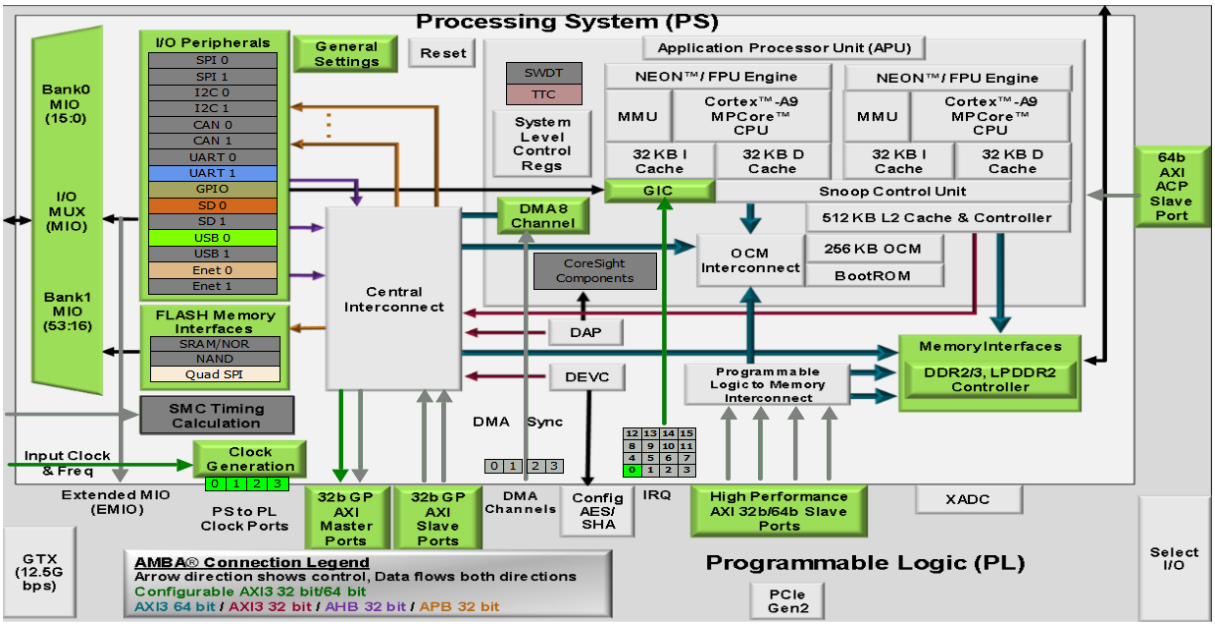
Po návrhu v matalbovském System Generatoru je dalším krokem pro vytvoření reálného obvodu spuštění Xilinx Platform studia. Přesněji jeho část, které se nazývá The Embedded Development Kit (EDK). Zde je možné najít a nastavit, na jaký portech a přes jaké součásti bude návrh oživen. Také se zde získají další informace a veškerá dokumentace. Dalšími částmi toho programu jsou **Xilinx Platform Studio (XPS) Tool Suite, Software Development Kit (SDK), Real-Time Operating System and Embedded OS Support** a **Processing IP and MicroBlaze Soft Processor Core.** [12]

„XPS“ obsahuje grafickou podporu pro příkazový řádek pro hardwarové platformy. Zahrnuje také inteligentní průvodce designu.

„SDK“ slouží jako návrhový prostředek pro tvorbu vestavěných aplikací. V tomto prostředí se programuje v programovacím jazyce C. Obsahem jsou také uživatelsky přizpůsobivé ovladače pro veškerou podporu Xilinx hardwaru. SDK obsahuje různé módy spouštění návrhu a také optimalizační algoritmy.[16]

Další částí EDK je „Real-Time operation System“. Jedná se o vývojový nástroj, který poskytuje balíček s nástroji od jiných výrobců pro práci v reálném čase. Poslední součástí je „Processing IP and Microblaze Soft Processor Core“, který slouží ke správnému určení propojení pinů pro jádro MicroBlazu.

Na Obr. 16 - Schematické hardwarové rozvržení ZedBoardu je zobrazeno, jak jsou jednotlivé součásti na desce propojeny.



Obr. 16 - Schematické hardwarové rozvržení ZedBoardu

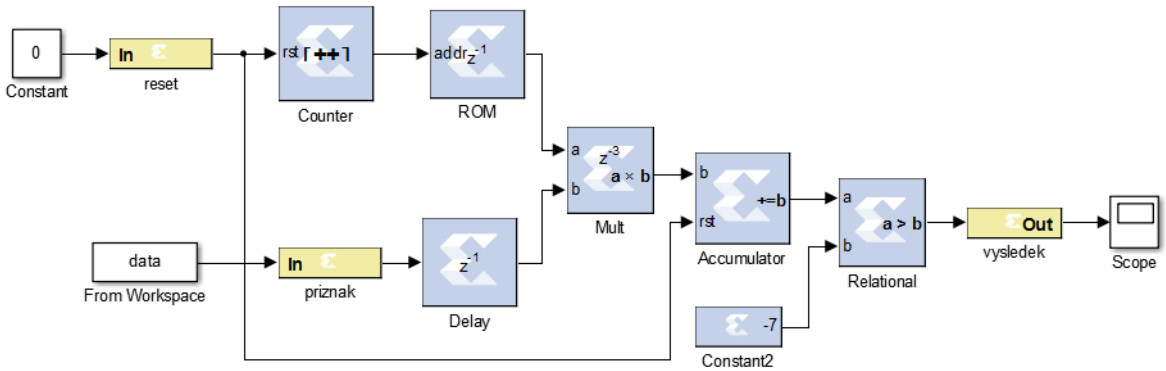
6. Neurony pro rozpoznávání znaků

Po určení a odsimulování neuronových sítí byla pro realizaci návrhu vybrána percetronová síť s příznakem převedení obrazu na vektor. Tato neuronová síť byla vybrána i přes to, že byla nalezena i lepší síť. Důvodem byla jednodušší realizace neuronových vstupů a výstupů a rozdíl ve výsledném rozpoznávání není takový.

Samotná realizace neuronu je pomocí jednoduchých bloků ze systém generatoru. Dále budou ukázány tři možné funční návrhy neuronů. Realizace celé sítě je poté provedena poskládání dostatečného počtu neuronů.

6.1 Jednoduchý neuron Percetronové sítě

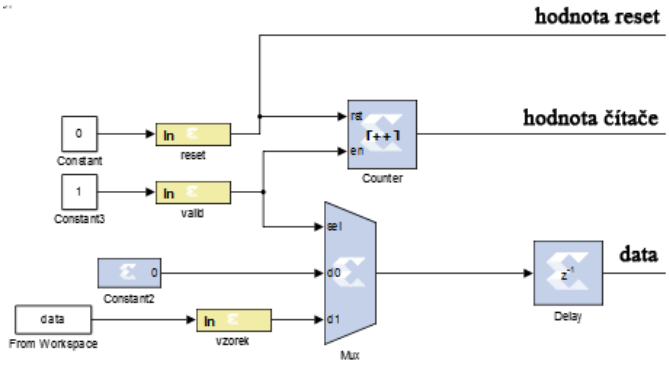
Jedná se o jednoduchý návrh. Doba provedení je 1024 taktů. Jelikož má obrázek se znakem velikost 32 x 32, má i celý příznak 1024 hodnot. Jedním ze vstupů je reset, který je použit z důvodu správného vymazávání čítače (Counter) a akumulátoru (Accumulator). Dalším vstupem je samotný zkoumaný znak ve formě dat. Tato data jsou dvourozměrným vektorem, v kterém je v první části uloženo číslo od 0 do 1023 a ve druhé je nositel informace. Hodnoty informace jsou ve velikosti jednoho bitu a jsou to tedy čísla jedna nebo nula. Čítač vysílá postupně čísla také od 1 do 1024 a díky tomu vychází z paměti ROM informace o vážené hodnotě příznaku pro určitý znak. Tyto hodnoty mají fixní velikost 8 bitů a jsou se znaménkem. Tedy rozsah od -127 do 127. Z paměti vycházejí data se zpožděním jedna, a proto se musí do větve s vedením dat přidat zpožďovač. Potom se již hodnoty spolu jen vynásobí a akumulují se v akumulátoru. Dochází k průběžnému porovnávání s hodnotou výstupního vektoru *b*. Na výstupu je poté hodnota jedna nebo nula. Jako výsledná informace se bere hodnota až v posledním taktu. Tento návrh je pouze základní a dále nebyl zkoušen. Jedná se o názorný příklad toho, jak neuron funguje. V příloze A na CD je uložen jako *„jednoduchy_neuron.slx“*.



Obr. 17 - Návrh jednoduchého neuronu

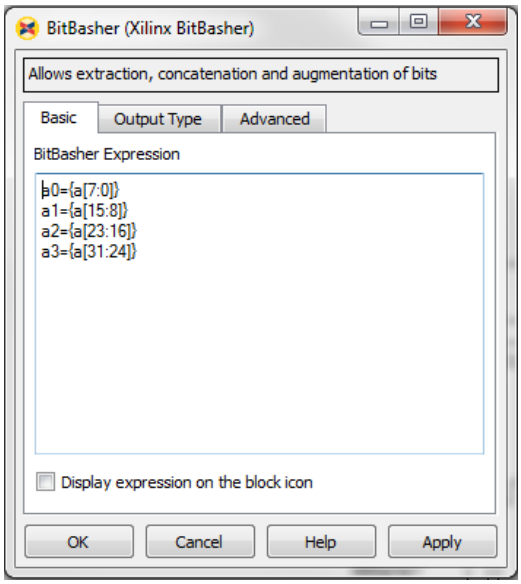
6.2 Neuron s Bitbasherem na 32 bitech

Další realizovaný neuron je čtyřikrát rychlejší ve výpočtu než jednoduchý neuron. To je 256 taktů. U tohoto neuronu jsou tři vstupy. Novým vstupem je hodnota *valid*, která data kontroluje a řídí, aby byla kompletní. Na Obr. 17. je ukázáno, jak jsou vstupy do neuronu řešeny a jejich ošetření.



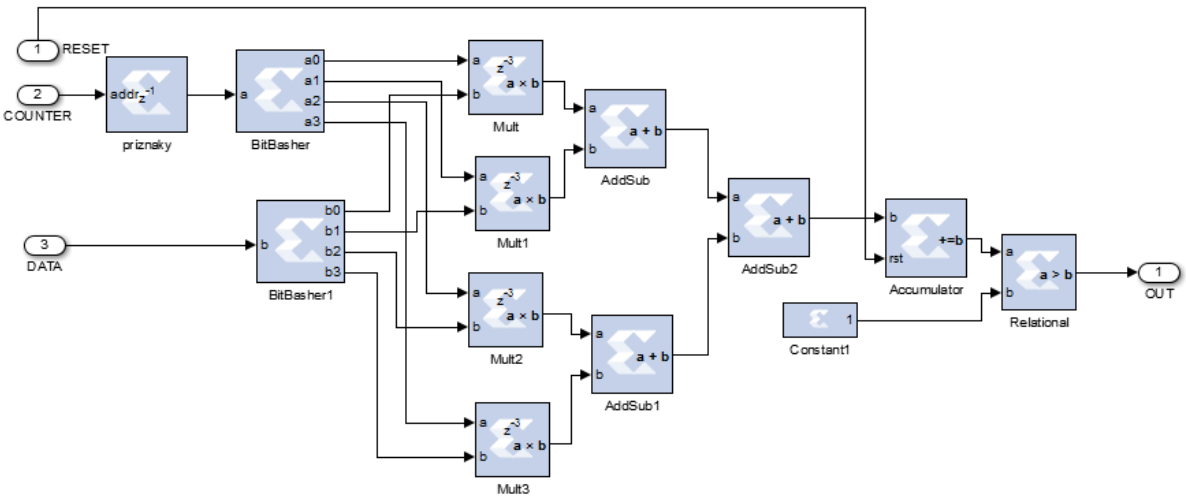
Obr. 18 - Ošetřený vstupy pro neurony

Samotný neuron je umístěn v samostatném subsystému. Princip je stejný jako u jednoduchého neuronu. Pouze vedená data jsou upravena blokem *Bitbasher*, který rozdělují informaci do požadovaných bitů. Příklad nastavení je na Obr. 18. Jeden vstup je rozdělen na čtyři výstupy a0 až a3. V závorkách je uvedeno, jaké bity jsou pro jaký výstup. Aby bylo toto možné, bylo nutné vstupní data upravit pomocí algoritmů, čtyři osmi-bitová čísla převedly jedno šedesáti-čtyř bitové. Ta samá úprava byla použita i na vstupu dat. Skripty pro přepočty jsou v příloze A na CD pojmenovány *„INPUTS_na_32.m“* a *„OUTPUST_na_32.m“*.



Obr. 19 - Ukázka nastavení Bitbasheru

Samotný neuron je na Obr. 19. Je zde vidět tři vstupy (RESET, COUNTER a DATA) a jeden výstup (OUT). Jeho vstupní data jsou rozdělena pomocí bitbasheru na čtyři samostatné datové cesty, které se spolu násobí. Aby vznikla pouze jedna informace pro akumulátor, musí se sečíst. Stejně jako u jednoduchého neuronu se výsledek porovnává s hodnotou vektoru *b*. Jako celkový výsledek se bere hodnota v posledním taktu na výstupu z neuronu. Tento návrh je v příloze A na CD uložen *„neuron_bitbasher_32“*. Celá síť je pojmenována *„neuron_sit_bitbasher_32“*.



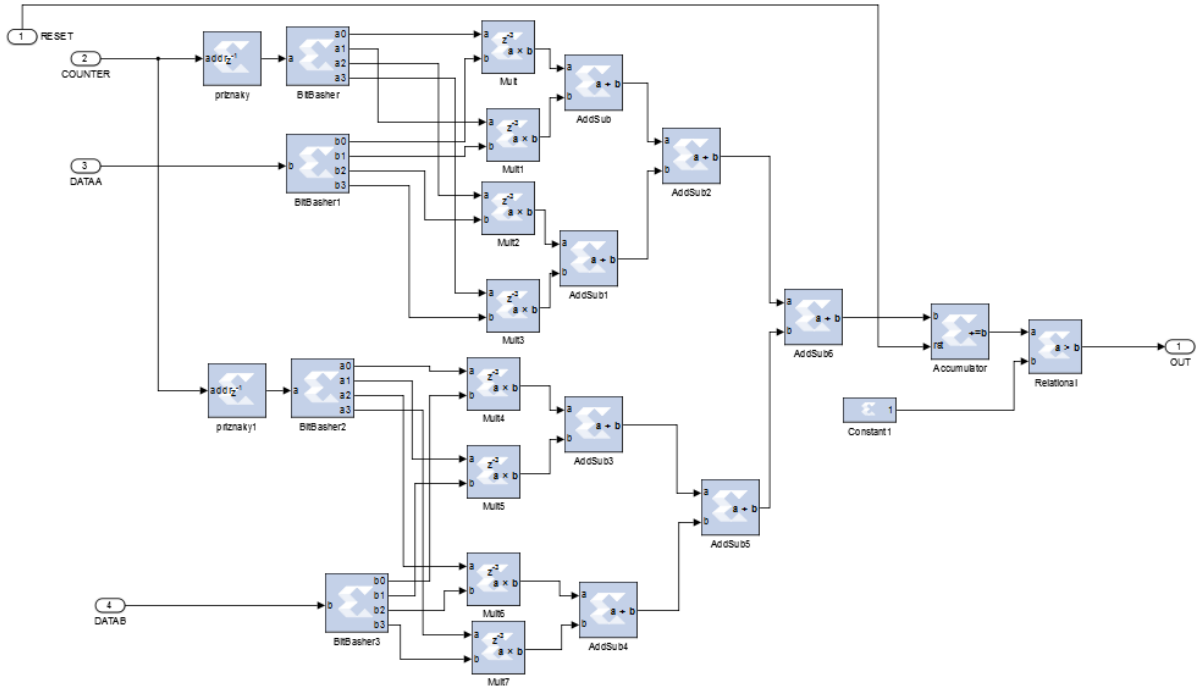
Obr. 20 - Návrh složitějšího neuronu s bitbasherem

Při tvoření sítě jsou výstupy z neuronu vyvedeny do bloku *Concat*, který z nul a jedniček vytvoří číslo. Rozpoznaný znak poté vychází tak, kde se na bitu nachází hodnota jedničky.

6.3 Neuron s Bitbasherem na 2x32 bitů

Další možností, jak zrychlit výpočet neuronu, je pomocí rozdělení vstupních dat na další dva kanály. To znamená dvě paměti a dva různé vstupy dat. Tento návrh je schopný rozpoznat znak za 128 taktů. Což je od původního jednoduchého neuronu osmi-násobné zrychlení.

Jako u předchozího návrhu jsou vstupy před samotným neuronem ošetřeny. Jsou zde dva vstupy dat *DATAA* a *DATAB*. Jedná se také od dvourozměrný vektor, ve kterém se v prvním sloupci nachází hodnoty od 0 do 127 a ve druhém samotná data přepočítána pomocí skriptu. Tento skript je stejný jako u předchozího příkladu, jen se poté data rozdělí. Pro paměti se jedná jen o správné napsání outF(1,1:2:256). U vstupních dat se hodnoty vytvoří takto dataA=[tt;dataa(1:2:256)']'; a dataB=[tt;dataa(2:2:256)']';. Je nutné je správě transponovat, aby byla data ve sloupcích vedle sebe. Proměnná *tt* je vektor od 0 do 127. S takto zpracovanými daty už jde dále pracovat a vytvořit fungující neuron. Postup výpočtu je stejný jako u předchozí verze, pouze se jedná o větší počet násobiček a sčítaček. Proto již další zrychlování rozdělením do více cest není přínosné, protože by pak celé neuronová síť obsahovala příliš mnoho násobiček.



Obr. 21 - Návrh složitého neuronu s dvěma vstupy

Při zapojení neuronů do sítě se jejich výstupy vyvedou do bloku *Concat*. Tento návrh je v příloze A na CD uložen *„neuron_bitbasher_2x32“*. Celá síť potom je pojmenována *„neuron_sit_bitbasher_2x32“*.

6.4 Hodnocení navržených sítí

Takto navržené sítě mají přesné parametry pro vstupní hodnoty a mají i přesné zapojení. Ve vnitřním zapojení je důležité hlídat, v jakém formátu se data pohybují a přepočítávají. Vytvořená data pro paměti ROM jsou specifická, protože v zapojení s bitbasherem mají výstup bezznaménkový, i když se jedná o budoucí záporná čísla. Toto zařídí rozdělení dle bitů. Přepočet znamének se provádí dvojkovým doplňkem.

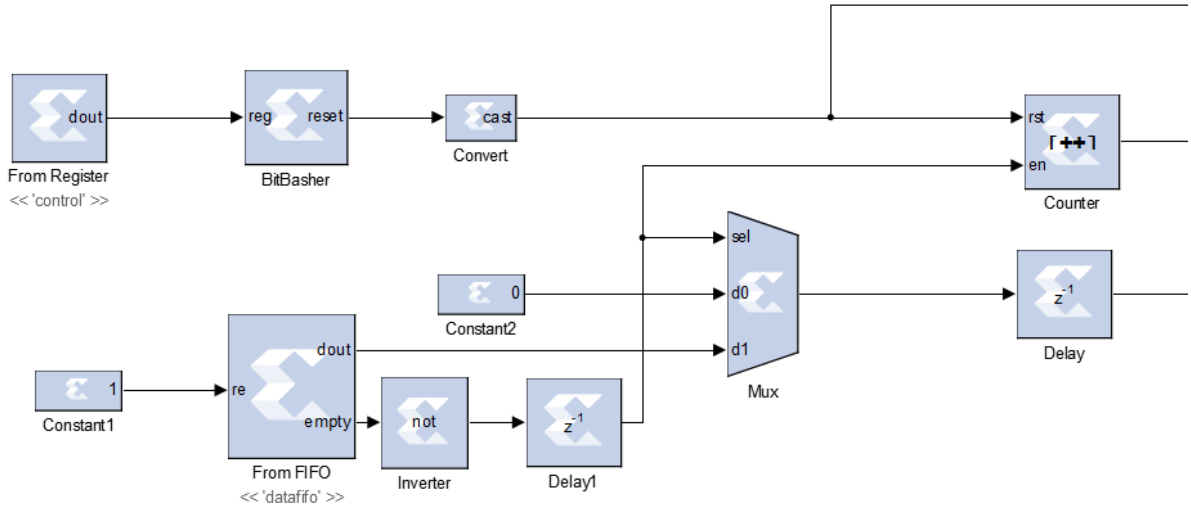
U všech navržených neuronových sítích bylo vše odsimulováno s mnoha různými znaky a byla sledována velikost nárůstu akumulátoru. Všechny výsledné hodnoty z akumulátoru byly stejné, lišily se pouze v časové ose, což je správně. V posledních okamžicích se již hodnota v akumulátoru ustálila, jelikož poslední části ve znaku jsou bílé plochy. Proto je výsledek znám většinou již po třech čtvrtinách výpočtů, ale je nutno ho dokončit celý z důvodů možné podobnosti některých znaků.

Výsledné počty taktů pro různé druhy rozpoznávání jsou: Pro jednoduchý neuron 1028, pro neuron s bitbasherem je to 256 taktů (čtyřikrát rychlejší). V posledním vytvořeném návrhu je osmkrát rychlejší než základní neuron tzn. 128 taktů. Přesnost všech sítí je stejná, protože mají totožný základ dat.

7. Reálné simulování na ZedBoard

Pro reálné odsimulování byl využit návrh *Neuron s Bitbasherem na 32 bitech*. Vstupy do samotného neuronu jsou stejné, ale jsou jinak vytvořené. Hodnota *valid* vzniká na speciálním výstupu paměti FIFO, který hlídá, zda jsou na výstupu paměti data. Pokud tato situace nenastane, je hodnota výstupu *empty* jedna. V paměti FIFO jsou uložena data zkoumaného příznaku. Další potřebný vstup je reset. Ten je vytvořen z informace, kdy je paměť s příznakem naplněna. Do obvodu bylo zaneseno další zpoždění. Jedná se o zpoždění čítače (Counteru), které bylo zjištěno až při simulacích. Celkové zpoždění na neuronové síti je 12taktů, které by se mělo započítat při zjišťování výstupní informace. Problém je ale s přístupem k této informaci. Zpoždění odezní dříve, než je možné se k informaci dostat.

Vytvoření nových vstupů je zobrazeno na Obr. 22. Tento návrh sítě je v příloze A na CD uložen *„final_neuron_bitbasher_32“*.



Obr. 22 - Reálný vstupy pro neuron

Do schématu byl také přidán blok procesoru (**EDK Processor**), který slouží pro porovnání výpočtu s hardwarovou verzí.

Dále pomocí bloku System Generator byl vygenerován kód pro platformu EDK, který byl uložen jako „system.xmp“. Všechny výsledné soubory jsou uloženy v příloze A na CD ve složce *„zynq_system“*.

Po spuštění Xilinx Platform Studia Xilinx Platform studio (EDK) se tento soubor otevírá jako nový projekt. Z načteného souboru se vytvoří tři adresy pamětí. Dvě vstupní, kterými jsou reset a data zkoumaných znaků a jedna výstupní adresa, do které se uloží výsledná informace. Adresy jsou deklarovány takto: pro výsledky je deklarována adresa *base 0x804*, pro řízení resetu je adresa *base 0x800* a pro vstupní data je adrese *base 0x400.* Po znalosti nutných adres je nutno zkontrolovat celkové využití desky a jejích částí. Z celkové kapacity FIFO pamětí bylo využito 10%. Z celkových 220 násobiček bylo využito 104, což je 47% využití. Celkové využití desky bylo 685 částí z 13300, což je 5% využití. Po kontrole a zjištění adres se návrh exportuje do programu Xilinx Software Development Kit (SDK).

V SDK jsou vytvořeny dvě rozdílné cesty řešení. Jedna je na připraveném hardwaru a druhá je softwarová za použití připraveného procesoru. Obě cesty vycházejí ze stejných dat. Pro první jmenovanou část byly využity předpřipravené knihovny. Proto je vytvořený program pouze čekání na poslední výslednou hodnotu a ta je brána jako výsledek. Ten je pak vypsán na připojené zobrazovací okno. Software využívá funkce také z předpřipravených knihoven. Výsledek je zde počítán postupně po jednom

neuronu. Všechny neurony se poté porovnají s připravenými hodnotami vektoru b. Poté se také vypíše hodnota znaku na stejný řádek jako hardwarové řešení. Během testu bylo všech cca 720 znaků nalezeno. Během všech výpočtů se počítá doba, za jak dlouho byl znak určen. Veškeré soubory k tomuto návrhu jsou uloženy v příloze A na CD ve složce *„zynq_system“*.

Nejprve budou uvedeny výsledky pro hardwarovou část. Rychlost určení znaku je 30,472 ms. Jedná se o optimalizovanou verzi. Pokud se vypne nástroj optimalizace, je doba určení znaku vyšší a to na 33,158 ms. Pokud by se zkoušela síť *Neuron s Bitbasherem na 2x32 bitů,* byl by výsledek dvakrát rychlejší. Tedy doba výpočtu by se pohybovala okolo 17 ms.

Výpočet softwarové části na procesoru byl pomalejší. Doba výpočtu byla 101,189 ms. Jednalo se o optimalizovanou verzi s využitím optimalizace O3. Výpočet bez zpuštěné optimalizace trval 691 ms. U toho řešení však nebylo využito maximálního potenciálu procesoru. Pro tento výpočet bylo použito pouze jedno jádro dvoujádrového procesoru. Při využití maxima výkonu procesoru by byla doba výpočtu okolo 50 ms. Pro tento specifický druh výpočtu by se dal také využít procesor NEON, který dokáže osmibitová data počítat čtyři najednou. Díky tomu by se výpočet zrychlil čtyřnásobně. Doba výpočtu by byla okolo 25 ms. Ale pro tento procesor se musí program speciálně přepsat.

Závěr

Tato diplomová práce řeší, jak je možné pomocí FPGA, přesněji na desce ZEdBoard provézt rozpoznávání znaků. V prvních částech je předvedena příprava a simulace v Matlabu před samotným návrhem obvodu. Jsou zde popsány programy, které vytvoří vzory pro každý znak. Vzorů bylo vytvořeno 8887 a musely být dále rozděleny. Počet vzorů pro každý znak je minimálně 100. Dále je každý vzorek optimalizován připraveným programem. V další části je popsáno vytvoření neuronové sítě s různými typy příznaků. Výsledky jsou porovnány a zhodnoceny do tabulky (Tabulka 2). Každá vytvořená neuronová síť byla testována pomocí sady 725 vzorků. Z výsledků byla určena pro další práce Percetronová síť, kde jako příznak byl brán „Převedení obrazu na vektor“. Důvod tohoto výběru je realizovatelnost a dobré procento úspěšnosti rozpoznávání. Z této sítě byly získávány hodnoty vážené sumy a informace meze výstupu pro každý neuron ke každému znaku. Poté začala příprava návrhu. V další části práce jsou popsány vlastnosti desky ZedBoard, pro kterou se bude navrhovat neuronová sít. Důležité bylo správné nastavení pamětí, vstupů a výstupů. Bylo provedeno několik variant návrhů. Postupnou aproximací od jednoduché realizace neuronu se dostáváme k finálnímu návrhu, který byl využit i na samotné desce ZedBoard. Doby výpočtu jednotlivých neuronů jsou ukázány v Tabulce 4. Pro samotnou realizaci byla použita síť s neurony s Bitbasherem na 32 bitech.

V poslední části práce navržená síť musela být ještě upravena, ale poté se již návrh z Matlabu pomocí prostředí Xilinx Platform Studio (EDK a SDK) převedl a otestoval na samotné desce ZedBoard. Výsledky využitých adres, komponent a porovnání časů jednotlivých výpočtů jsou zobrazeny v Tabulce 4.

Cílem do budoucna by bylo možné pokusit se využít více výpočetního výkonu desky. Jako další možnost rozšíření a zpřesnění rozpoznávání by bylo před celkový výstup dát slovník, který by z příchozích písmen vybral slovo s největší pravděpodobností. Díky tomu by se určitě zvýšila přesnost rozpoznávání.

Tabulka 4 - Celková naměřená a vytvořená data

Příznak - pro síť Percetronu	**Doba výpočtu**	**Počet správně poznaných znaků**	**Procenta úspěšnosti**
2.2.1 Převedení obrazu na vektor	5-7hod	708 z 725	97,70%
Navržené sítě		**Doba výpočtu v taktech na jeden neutron**	
Jednoduchý neuron percetronové sítě		1024	
Neuron s Bitbasherem na 32 bitech		256	
Neuron s Bitbasherem na 2x32 bitech		128	
Reálné hodnoty při nahrání obvodu do ZedBoardu			
Celkové zpoždění sítě		12 taktů	
Adresa vstupu dat		base 0x400	
Adresa řízení (reset)		base 0x800	
Adresa výstupu		base 0x804	
Využití pamětí FIFO		10%	
Využití násobiček		104 z 220	47%
Celkové využití desky		685 částí z 13300	5%
Výpočet na HW		30,472ms	s optimalizací
Výpočet na HW		33,158ms	bez optimalizace
Teoretický výpočet HW při použití sítě s Neuron s Bitbasherem na 2x32 bitů			17ms
Výpočet na SW		101,189ms	s optimalizací
Výpočet na SW		691ms	bez optimalizace
Využití celé kapacity procesoru		50ms	odhad
Využití procesoru NEON		25ms	odhad

Použitá literatura

[1] BISHOP, Christopher M.: *Pattern Recoginition and Machine Learning*, Springer, 2006, ISBN: 0-387-31073-8

[2] DUŠEK, František. *MATLAB a SIMULINK: úvod do používání*. Vyd. 1. Pardubice: Univerzita Pardubice, 2000, 146 s. ISBN 80-719-4273-1.

[3] Grupo ARCO. *How Use the Tree AXI Xonfigurations* [online]. 2009 [cit. 2014-05-16]. Dostupné z: https://arco.esi.uclm.es/public/doc/tutoriales/Xilinx/old_training/Embbeded%20Designs/3-axi-configurations.ppt

[4] CHALUPNÍK, Vitalij. Biologické algoritmy: Neuronové sítě. *Root.cz (www.root.cz), informace nejen ze světa Linuxu* [online]. 2012, č. 4 [cit. 2014-05-12]. Dostupné z: http://www.root.cz/clanky/biologicke-algoritmy-4-neuronove-site/#ic=serial-box&icc=text-title

[5]CHYTKOVÁ, Dagmar. *Vývoj písma* [online]. 2005 [cit. 2014-05-12]. Dostupné z: http://www.phil.muni.cz/~dchytkova/

[6] KRIESEL, David: *Neural Networks, a brief introduction to*, [online]. 2005 [cit. 2013-10-13]. Dostupné z: http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf

[7] KUTHAN, Vít. *Vývojový modul s programovatelným logickým polem FPGA*. Plzeň, 2013. Dostupné z: https://otik.uk.zcu.cz/handle/11025/9210. Diplomová práce. Západočeská univerzita v Plzni.

[8] MEDLÍK, Milam. *Rozpoznávání ručně psaných symbolů*. Univerzita Tomáše Bati ve Zlíne, 2010. Dostupné z: http://dspace.k.utb.cz/bitstream/handle/10563/11800/medl%C3%ADk_2010_dp.pdf?sequence=1. Diplomová práce. Fakulta aplikované informatiky.

[9] MOLNÁR, Karol. Úvod do problematiky umělých neuronových sítí. [online]. 2000 [cit. 2014-05-12]. Dostupné z: http://www.elektrorevue.cz/clanky/00013/index.html#typy

[10] *Neuron* [online]. 2006 [cit. 2014-05-16]. Dostupné z: http://ms.gymspgs.cz:5050/bio/Images/Textbook/Originals/%5B0090000%5D%20Nervov%C3%A1%20soustava/%5B00340%5D%20Neuron%20%5BE%5D.jpg

[11] PINKER, J., Poupa, M.: *Číslicové systémy a jazyk VHDL*, BEN, 2006, ISBN: 80-7300-198-5

[12] *Platform Studio and the Embedded Development Kit (EDK)* [online]. 2014 [cit. 2014-05-15]. Dostupné z: http://www.xilinx.com/tools/platform.htm

[13] ŠÍMA, Jiří. *Teoretické otázky neuronových sítí*. Vyd. 1. Praha: MATFYZ press, 1996, 390 s. ISBN 80-858-6318-9.

[14] ŠVEC, Zdeněk a Jakub ŽLÁBEK. *Systémy OCR* [online]. České vysoké učení technické v Praze, 2010 [cit. 2014-05-12]. Dostupné z: http://geo3.fsv.cvut.cz/vyuka/kapr/sp/2012/pokorny/index.html. Semestrální práce v Praze

[15] VIKTORIN, Jan. *HW/SW CO-DESIGN FOR THE XILINX ZYNQ PLATFORM* [online]. Brno, 2013 [cit. 2014-05-12]. Dostupné z: http://www.diplomovaprace.cz/118/14453.pdf. Diplomová práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ.

[16] *Xilinx Software Development Kit (SDK)* [online]. 2014 [cit. 2014-05-15]. Dostupné z: http://www.xilinx.com/tools/sdk.htm

[17]*ZedBoar* [online]. 2014 [cit. 2014-05-16]. Dostupné z: http://www.zedboard.org/

[18] *Zynq-7000 All Programmable SoC Overview* [online]. 2013 [cit. 2014-05-12]. Dostupné z: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

Příloha A – přiložené CD

Přiložené CD obsahuje:

- Diplomovou práci v elektronické podobě.
- Všechny zmíněné algoritmy a návrhy.
- Výsledný návrh pro desku ZedBoard.